(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number: 0 304 146 A2

(12) EUROPEAN PATENT APPLICATION

(21) Application number: 88305545.1

(22) Date of filing: 17.06.88

(51) Int. Cl.4: G06K 7/14

(30) Priority: 18.06.87 US 64110

(43) Date of publication of application: 22.02.89 Bulletin 89/08

(84) Designated Contracting States: BE DE FR GB IT LU NL SE

(71) Applicant: Spectra-Physics, Inc. (a Delaware corp.)
3333 North First Street
San Jose, California 95134-1995(US)

(72) Inventor: Cherry, Craig Douglas
3320 Videra Drive
Eugene Oregon 97405(US)
Inventor: Taussig, Andrew Peter
2515 West 22nd Avenue
Eugene Oregon 97405(US)
Inventor: Brooks, Michael T.
25112 Territorial Court
Veneta Oregon 97487(US)

(74) Representative: Bankes, Stephen Charles Digby et al
BARON & WARREN 18 South End Kensington
London W8 5BU(GB)

(54) Method of decoding a binary scan signal.

(57) A method of decoding a binary scan signal consisting of a bit sequence produced by an electro-optical scanning device (10) as the device (10) scans bar code symbols on a label is disclosed. The bits in the sequence correspond to light and dark spaces making up the bar code symbols. The method includes the steps of: a.) supplying the binary scan signal to a storage buffer (12) such that the buffer (12) contains a plurality of bits most recently produced by the scanning device (10), b.) selecting a portion of the bit sequence which defines a large light space, c.) subjecting the bits in the sequence following those defining the large light space to a series of tests to determine whether such bits were produced by scanning a bar code symbol which is valid in one or more of several bar codes, d.) decoding the bar code symbol in the codes in which it is valid, e.) subjecting the next bits in the sequence to a series of tests to determine whether such bits were produced by scanning a bar code symbol which is valid in any of the bar codes in which the previously decoded bar code symbol is valid, f.) decoding the bar code symbol in the codes in which it and the previously decoded bar code symbol are valid, and g.) repeating steps f.) and g.) above until all bar code symbols on the label have been decoded.

EP 0 304 146 A2

METHOD OF DECODING A BINARY SCAN SIGNAL

The present invention relates to bar code scanners and, more particularly, to label scanning systems of the type which are designed to read labels having information which may be presented in any of a number of different codes commonly in use.

Labels bearing information in a number of different bar code formats are typically used in a number of different applications. It is common, for example, to encode product identification information as a series of printed bars or dark areas, and interposed white spaces or light areas, on a product package or on a label affixed to the product package. An electro-optical scanner located at a check-out station in a retail establishment is then used by a clerk to automatically enter the product identification data into a computer terminal. This permits the computer to then determine the total price of the products being purchased, as well as storing the identity of the purchased products for inventory and accounting purposes.

While such an arrangement greatly enhances the efficiency of the check-out process in the retail establishment and additionally allows the accumulation of sales data which is important for proper management controls, difficulties are encountered in the scanning operation due to the nature of the products being scanned and the number of different bar codes currently in use. Packaging for various products is designed to make the products appealing to the consumer and, as a consequence, may include various graphics and text which, when scanned, produce a binary scan signal which mimics that produced when a label having valid bar code symbols is scanned. Additionally, a number of different bar codes have come into popular use, and the scanner circuitry must be capable of recognizing and decoding labels printed in each of these codes.

It is important that the scanner system be capable of accomplishing these tasks automatically, without intervention by the clerk, even if several labels in different bar codes are affixed to a single product. This presents substantial difficulties since the bar codes vary significantly in their formats. As an example, Code 3 of 9 is a binary code using elements of two widths in a symbol to represent a single character. Each of the 44 defined patterns of bars and spaces consists of five bars and four spaces. Each pattern represents one character in the forward direction and has the appearance of a second character in the reverse direction. The Interleaved 2 of 5 code, on the other hand, is a binary code using elements of two widths to represent numeric characters. Each frame or symbol, ten elements in length, contains two characters, the first being represented by the bar pattern, and the second by the space pattern. There is no gap between adjacent characters. In both codes, the proper scan direction, which may be a direction opposite to that in which the symbol was actually scanned by the electro-optical scanner, is determined by start and stop patterns at the beginning and end of the string of characters.

A number of other bar codes have also come into common use, including for example Codabar, Code 93, Code 128, the Universal Product Code (UPC), and the European Article Numbering (EAN) code. It will be appreciated that there is a need for a method of decoding a label in any of these codes without requiring an operator assessment of the specific code used for the label.

This need is met by a method according to the present invention for decoding a binary scan signal consisting of a bit sequence produced by an electro-optical scanning device as the device scans bar code symbols on a label, with the bits in the sequence corresponding to light and dark spaces making up the bar code symbols. The method comprises the steps of: a.) supplying the binary scan signal to a storage buffer such that the buffer contains a plurality of bits most recently produced by the scanning device; b.) selecting a portion of the bit sequence which defines a large light space; c.) subjecting the bits in the sequence following those defining the large light space to a series of tests to determine whether such bits were produced by scanning a bar code symbol which is valid in one or more of several bar codes; d.) decoding the bar code symbol in the codes in which it is valid; e.) subjecting the next bits in the sequence to a series of tests to determine whether such bits were produced by scanning a bar code symbol which is valid in any of the bar codes in which the previously decoded bar code symbol is valid; f.) decoding the bar code symbol in the codes in which it and the previously decoded bar code symbol are valid; and g.) repeating steps e.) and f.) above until all bar code symbols on the label have been decoded.

The several bar codes may include one or more codes selected from the group consisting of Code 3 of 9, Interleaved 2 of 5, Codabar, Code 93, Code 128, and UPC/EAN, or others using similar data encoding methods.

One of the series of tests may be the comparison of the element ratio of the bits being tested with preset minimum and maximum element ratio levels, the element ratio being the ratio of the width of the widest of the dark spaces making up the symbol to the width of the narrowest of the dark spaces making up the symbol.

One of the series of tests may be the comparison of the element ratio of the bits being tested with preset minimum and maximum element ratio levels, the element ratio being the ratio of the width of the widest of the light spaces making up the symbol to the width of the narrowest ofthe light spaces making up the symbol.

One of the series of tests may be the comparison of the margin ratio of the bits being tested with a preset minimum margin ratio level, the margin ratio being the ratio of the width of the large light space preceding the symbols on a label to the sum of the width of the first several light and dark spaces making up the first symbol adjacent the large light space.

One of the series of tests may be the comparison of the threshold ratio of the bits being tested with a preset with a preset threshold ratio, the threshold ratio being the ratio of the width of the widest light or dark space making up the symbol to the width of a particular light or dark space within the symbol.

One of the series of tests may be the comparison of the character ratio of the bits being tested with preset maximum and mimimum character ratio levels, the character ratio being the ratio of the sum of the widths of the light and dark spaces making up a symbol to the sum of the widths of the light and dark spaces making up the previous symbol.

One of the series of tests may be the comparison of the gap ratio of the bits being tested with preset maximum and mimimum gap ratio levels, the gap ratio being the sum of the widths of the light and dark spaces making up a symbol to the width of the light space between the symbol and an adjacent symbol.

One of the series of tests may be the comparison of the maximum narrow element ratio of the bits being tested with a preset maximum narrow element ratio level, the maximum narrow element ratio being the greater of the maximum ratio of the width of the narrowest dark space in a symbol to the width of the narrowest light space in the symbol, or the maximum ratio of the width of the narrowest light space in the symbol to the width of the narrowest dark space in the symbol.

Step d.) may include the step of checking to determine whether the decoded bar code symbol is a backward or forward start or end symbol prior to subjecting further bits in the sequence to testing and decoding.

Step g.) may include the step of checking the decoded bar code symbol to insure that it is decoded as a symbol which is one of a valid set of such symbols prior to g steps e.) and f.).

The method may further comprise the step of comparison of the margin ratio of the bits in the sequence the final symbol with a preset minimum margin ratio level, the margin ratio being the ratio of the width of a large light space following the symbols on a label to the sum of the width of the last several light and dark spaces making up the last symbol adjacent the large light space.

At least some of the tests to determine whether the bits in the bit sequence were produced by scanning a bar code symbol which is valid in several codes may be performed simultaneously. Alternatively, the tests to determine whether the bits in the bit sequence were produced by scanning a bar code symbol which is valid in several codes may be performed sequentially.

Steps a.) through g.) may be performed by a programmed digital computer.

Accordingly, it is an object of the present invention to provide a method of decoding a binary scan signal consisting of a bit sequence produced by an electo-optical scanning device as the device scans bar code symbols; to provide such a method in which the bar code symbols are automatically decoded regardless of which of several different codes are scanned; and to provide such a method in which the bar code symbols are not decoded in an erroneous bar code.

In order that the invention may be more readily understood, an embodiment will now be described, by way of example, with reference to the single Figure which is a schematic representation of a scanner, storage buffer, and microprocessor which may be utilized to perform the method of the present invention.

The present invention relates to a method for decoding a binary scan signal consisting of a bit sequence produced by an electro-optical scanning device as the device scans bar code symbols on a label. The bits in the sequence correspond to the widths of the light and dark spaces defined by the spaces between the bars and by the bars themselves, respectively. Detailed algorithms for decoding the referenced bar codes, given measurements of the bars and spaces in a label, are provided below. The preferred apparatus by which this method is implemented is a programmed microprocessor. A hardware based decoder system may, however, use the same label acceptance criteria, with suitable use of parallelism and some adjustment of the numeric ratios and limits for ease of computation.

The goal in developing the algorithms utilized in the present invention was low error rate, fast response, and compatibility with autodiscrimination of the various bar codes. Low error rate requires that all available information in the measurements be used in deciding if a good label is present. The measurements and methods used eliminate the effects of systematic errors in the printing and scanning process. Fast response while autodiscriminating several codes requires efficient implementation and early recognition of being in

the wrong code. Various data capture methods may be supported, such as capture of a scan before decoding, or concurrent data capture and decoding.

The alogorithms operate according to the following specific guidelines:

A. All bars and spaces in the label are checked for validity in some way.

B. Label margins are required.

C. Labels may be read forward or backward.

D. Tests for label validity are done sequentially, starting with the fastest screen for a good label, progressing to the tests which generally reject fewer labels, or take longer to compute. Details of these tests are given in the sections for each code in this document. Failure of any test during this process results in trying a different decoder algorithm, or looking for a label at a different point in the scan data.

E. After a label appears valid based on its structure and analysis of its element widths, additional operations may be performed, such as checksum validation and postprocessing into the final data to be sent from the decoder. Efficiency of the scanning and decoding process depends on the control structures used to look for labels and call the software decoders.

Referring to the figure, the algorithms disclosed herein assume that a binary scan signal, consisting of a bit sequence produced by an electro-optical scanning device 10, is available in a buffer 12. This bit sequence is produced by the scanning device 10 as it scans a package bearing a bar code label. Buffer 12 contains a complete scan pattern or a moving window of the scan data. Potential locations for the start of a symbol on the label are identified by checking for large white spaces defined by the data, which may correspond to the label margin. (It should be understood that throughout this discussion, "white space" and "light space" may be used interchangeably, since some bar code patterns may not be printed on truly white background surfaces. Similarly, the "bars" may also be referred to as "dark spaces.") Decoder algorithms, performed by microprocessor 14, preferably a Thompson-Mostek MK68HC200 microprocessor, are then accessed to attempt decoding in the particular codes at the buffer location specified. It will be appreciated that other microprocessors may also be utilized, if desired.

Each decoder algorithm specified will try to decode a label at the location specified. If a good label is not found, other decoder algorithms may then be used for the same data until all have been tried. If none are successful, the buffer may then be searched for another possible label. The buffer may be filled during one sweep of the scan pattern, then decoded, or data may be added to the buffer continuously and decoded concurrently. While the various algorithms are accessed sequentially in the preferred embodiment, it will be appreciated that the selected algorithms may be accessed simultaneously. The tests by which valid bar code symbols may be recognized for the various codes may be performed prior to decoding. Alternatively, a portion of the tests may be performed, preferably the tests which may be performed quickly, and then the symbols decoded. The balance of the tests may then be performed. A complete hardware implementation of the present invention would utilize similar decisions in the actual decoding process. The algorithms by which the method of the present invention are effected are given in the following sections, delineated Sections 4 through 9.

Section 4: Code 3 of 9

4.1 General Description of Code 3 of 9

See the referenced document for details. This is a binary code using elements of two widths to represent alphanumeric and some special characters. Each character pattern contains three wide elements and six narrow elements. The ratio of wide elements to narrow elements may vary over the range of 2:1 to 3:1 from one label to another, but is to be constant within a given label. There is a gap between adjacent characters. Each of the 44 defined bar/space patterns consists of 5 bars and 4 spaces. Each pattern represents one character in the forward direction and looks like another character in the reverse direction. The actual scan direction is determined by looking for one of two possible valid characters at the beginning of the label: the usual start/stop character, or a reversed version of the start/stop character. In order to reduce errors due to systematic distortion of bar and space size, the decoding process for each character treats the bars and spaces separately. The number of data characters in a label is variable. A maximum length of 32 data characters is commonly specified. Optional features including label concatenation, a check character, and 128 character ASCII character set are defined in the AIM spec.

4.2 Overview of the algorithm.

Variables and constants used in the decoding process are defined below, followed by a brief descriptiojn of the decoding algorithm. A detailed description of the algorithm follows in section 4.3.

4.2.1 Status Variables

Values of the status variables can be maintained globally to allow re-entrant use of the decoder.

Variable name	Description
i	pointer to the current element in the data buffer. This always points to a space when the decoder is called.
last char width	sum of the element widths in the last character decoded in the current label. This does't include the intercharacter gap.
current character	decoded ASCII character
label string	decoded characters as an ASCII string.
forward	boolean; true if decoding in forward direction.
continue decode	boolean; true while the buffer at the current element looks like good data.
found label	boolean; true when a label has been put into label string.
data buffer	array of numbers representing widths of the elements scanned. They must alternate between bars and spaces.

4.2.2 Decode Constants

Constant values are identified in this section. The constants are referenced by name in the description of the algorithm

Constant name	Value	Description
frame width	10	Number of elements in a character plus the gap.
threshold ratio	.70	Ratio of the width of the widest bar (space) in a character to the wide/narrow decision point.
min element ratio	1.5	lower limit on the ratio of the width of the widest bar (space) to the narrowest bar (space) in a character.
max element ratio	5.0	upper limit on the ratio of the width of the widest bar (space) to the narrowest bar (space) in a character.
max narrow element ratio	3.0	max ratio of the narrowest bar in a character to the narrowest space, or vice versa.
max char ratio	1.25	max ratio of the sum of the elements in the current character to last char width, not including the intercharacter gap.
min char ratio	.80	min ratio of the sum of the elements in the current character to last char width, not including the intercharacter gap.
max gap ratio	30	max ratio of the sum of the elements in the current character to the previous intercharacter gap.
min gap ratio	2.0	min ratio of the sum of the elements in the current character to the previous intercharacter gap.
min margin ratio	1.0	min ratio of the width of the white space before the label to the sum of the width of the first four elements of the label.

4.2.3. Derivation of constants

threshold ratio

This is chosen to be near the midpoint of the difference between a narrow element and a wide element. For an N:1 wide/narrow ratio the value giving the midpoint is t=(((N-1)/2)+1)/N. For N=3 t=.6666; N=2.5 t=.7000; N=2 t=.7500. We are using .7000. If an application only used a fixed ratio other than 2.5:1 this could be optimized to match. If using a simple integer ratio is helpful t=23/32 corresponds to N=2.3. If labels are rejected due to a wide variation in the width of the narrow elements in the four characters $/+% (because of the test for all narrow elements) the ratio could be decreased.

min element ratio

This is set to one half of the 2:1 minimum spec value for the wide/narrow ratio. Again, it and other limits could be changed for a particular application.

max element ratio

This is set to 5.0 based on the ratio of the widest wide element to the narrowest narrow element when the spec tolerances are applied to a nominal label. The actual worst case ratio is (3*.040+.014)/(.040-.014)=5.15.

max narrow element ratio

This is the only comparison other than total character width limiting the size of bars versus spaces. This is based on the spec ratio of a narrow element plus the tolerance to a narrow element minus the tolerance, plus some more to allow for scanning errors and printing errors beyond spec. The limit per spec is: (.040+.014)/(.040-.014)=2.08. Use 3.0. There is no theoretical upper limit to this, but it shouldn't be set to a value larger than will be seen in a label that is otherwise intact enough to decode.

max char ratio, min char ratio

These are chosen based on the possible scanning spot velocity variation within a character and the scanning and printing error over the character elements. Use 1.25 and .80 until better data for the particular scanning device being used is available.

max gap ratio

This is based on the ratio of the sum of the width of the elements in the widest legal character (excluding the gap) to the narrowest gap. In a label with a 3:1 wide/narrow ratio a character is 15 nominal elements wide. At .040 inch nominal element size the minimum gap is .040-.014 or .65 nominal. This gives a ratio of 15/.65=23. Use 30 to allow for out of spec gap widths.

min gap ratio

This is based on the ratio of the sum the width of the of the elements the narrowest legal character (excluding the gap) to the widest gap. In a label with a 2:1 wide/narrow ratio a character is 12 nominal elements wide. The widest gap is 5.3 times a nominal element per spec, for a ratio fo 12/5.3=2:26. Use 2 to allow some extra margin for error.

min margin ratio

This is a compromise between enforcing the rigorous spec limits and program efficiency. The minimum white space per spec is 10 times the nominal element size. The sum of the first four elements of the label for a 2:1 label is 5 times nominal; for a 3:1 label it is 6 times nominal. This results is a min margin of 5 to 6 elements, which allows for out of spec labels and scanning error.

4.2.3 General Decoding method

For efficiency the data buffer should be searched for a large white space indicating a potential label margin. Then all decoders can start processing from this point, avoiding duplicating the search process. If the decoder doesn't find a good label starting at this position it will exit back to the calling program. Before the decoder is first called some of the status variables must be initialized. They should be set as follows:

i : set to point to large white space.

Before any operation that looks at the data buffer, it is assumed that proper care will be taken to make sure that data is available in the buffer.

Each time the decoder is called it makes the comparison specified by min margin ratio. If this test passes, it looks for a forward or backward start character pattern and tests the elements in the character using threshold ratio, min element ratio, max element ratio, and max narrow element ratio. If this is all ok, the following variables are set:

continue decode : true
forward : true or false, depending on the character found

last char width : set to the sum of the elements in the start character
label string : set to empty
i : set to the current value of i + frame width.

Then it continues going through the label elements, appending characters to the label string until an error occurs, or the end character is found. For each character, the same checks made for a start character are applied (except the min margin ratio) plus the intercharacter checks for max char ratio, min char ratio, max gap ratio and min gap ratio. If the character found wasn't a stop character and all tests passed, the status variables are updated:

last char width : set to the sum of the elements in the character just found
label string : the character found is appended to label string
i : set to the current value of i plus frame width

If any test fails, continue decode is set false.

If the character was a stop character, set continue decode false, and check for the trailing margin using min margin ratio. If ok, do any secondary processing such as reversing the label string to correct for a backward scan, evaluating an optional check character, expansion to full ASCII, label concatenation, etc. Then if everything is ok set found label true.

The check for a good character pattern is done by finding the widest bar and space in the character, multiplying each by the threshold ratio, then comparing the result to each bar and space in the character to identify the wide and narrow elements. Note that this treats bars and spaces independently. The result is recorded as two binary patterns, one for bars and one for spaces, with ones indicating wide elements. Since it is possible for a Code 3 of 9 character pattern to have no wide bars (which would look like all wide bars to the above procedure), a separate test must be performed to detect this and correct the binary pattern. The two binary numbers are used to find table entries indicating if a good character was read and its value. If a good character pattern was found, the smallest bar and space, and the total character width are determined. Tests for min and max element ratios are done independently for bars and spaces, taking into account the case of all narrow bars. The max narrow element ratio tests limits the difference between the width of bars and spaces. If all these tests are ok, a good character was found.

4.3 Code 3 of 9 Decode Algorithm

The decoding algorithm is given below. If any label integrity test fails, an exit from the algorithm will occur with the status variable found label set to false. Before calling the decoder set i to point to a possible margin (wide white space). Information in the scan data buffer is refered to as element(I) for the Ith element of the data buffer. During the decoding process i is assumed to point to a margin or intercharacter gap. Follow the steps as specified, starting at 4.3.1.

4.3.1 Set found label false.

4.3.2 If less then frame width counts are available to be examined wait. + frame width against the last buffer location.)

4.3.3 If (element(i) < min margin ratio * (the sum of element(i+1) through element(i+4)) quit (margin too small).

4.3.4 Do the steps 4.3.11, 4.3.12, and 4.3.13 to look for a start pattern. If the bar pattern found is 00110 and the space pattern found is 1000 set forward true, or if the bar pattern is 01100 and the space pattern is 0001 set forward false. If neither combination was found quit (no start char found). Otherwise do the procedure specified starting at step 4.3.17 to check the element widths in the character. If they don't pass the tests, quit (character elements out of limits). Otherwise set continue decode true, set last char width to the sum of the elements in the character computed at step 4.3.17, set label string to empty, and increment i by frame width. An apparent label start has been found.

4.3.5 If less then frame width counts are available to be examined wait. (Check i + frame width against the last buffer location.)

4.3.6 Do the procedure specified starting at step 4.3.11 to get the character pattern. If a legal pattern wasn't found set continue decode false and quit (no char found). Otherwise do the procedure starting at step 4.3.17 to check the element widths in the character. If they don't pass the tests, set continue decode false and quit (character elements out of limits).

4.3.7 Compute the ratio of the sum of the elements in the current character to last char width. If this ratio is greater than max char ratio or less than min char ratio set continue decode false and quit.

4.3.8 Compute the ratio of the sum of the elements in the current character to element(i). If this ratio is greater than max gap ratio or less than min gap ratio set continue decode false and quit.

4.3.9 If the current character found in step 4.3.6 is '*' go to step 4.3.10. Otherwise if the length of label string is the maximum allowable label length set continue decode false and quit (label string overflow). Otherwise set last char width to the width of the current character, add frame width to i, and append the current character to label string. Go to step 4.3.5.

4.3.10 Set continue decode false. If element(i + framewidth) < min margin ratio * (the sum of element-(i + 6) through element(i + 9)) then quit. Otherwise if forward is false reverse label string, and do any optional operations for check sum, full ASCII character set, or concatenation (per the referenced spec). If no errors are found set found label true and quit.

4.3.11 This section is referenced from the main flow of the algorithm and you should return to the step which called this one when done here. Only the first three steps (11-13) will be done when looking for a start character. The elements of the current character will be examined looking for a good character pattern. Find the widest bar and widest space in the nine elements i + 1 through i + 9. Multiply each of these by threshold ratio to find the wide/narrow breakpoint.

4.3.12 Set a binary number which will represent the bar pattern to 0. Now for each of the elements i+1, i+3, i+5, i+7, and i+9, multiply bar pattern by 2, then increment it by 1 if the element under consideration is greater than the bar threshold calculated in step 4.3.11. When done if bar pattern equals 11111 binary then set bar pattern to 0 (no real character has all wide bars, but some have all narrow, which requires this adjustment).

4.3.13 Set a binary number which will represent the space pattern to 0. Now for each of the elements i+2, i+4, i+6, and i+8, multiply space pattern by 2, then increment it by 1 if the element under consideration is greater than the space threshold calculated in step 4.3.11.

4.3.14 If bar pattern isn't 0 go to 4.3.15. Otherwise set char pointer according to the value of space pattern:

```
space pattern:  7   char pointer: 44
               11                 43
               13                 42
               14                 41.
```

If space pattern isn't one of these four values return; the test failed. Otherwise go to 4.3.16.

4.3.15 Use the two look up tables
space index[0..15] = 0,3,2,0,1,0,0,0,4,0,0,0,0,0,0,0
bar index[0..31] =
0,0,0,7,0,4,10,0,0,2,9,0,6,0,0,0,0,1,8,0,5,0,0,0,3,0,0,0,0,0,0,0
to determine what character is represented by the bars and spaces. If bar index[bar pattern] = 0 or space index[space pattern] = 0 then return; the test failed. Otherwise set char pointer to
10* (space index[space pattern]-1) + bar index[bar pattern].
This process takes advantage of the repetitive bar and space patterns used in the Code 3 of 9 characters.

4.3.16 Use char pointer to select a forward character or backward character from these two lists of 44 characters. For example, if char pointer is two and forward is false, pick the second character from the backward list, and so on. If forward is true, set current character to the correct character in the forward character list, otherwise use the backward character list.
Forward characters:
1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZ-. *$/+%
Backward characters:
AHGEDJCBIF1875403296U. -YX*WV ZKRQONTMLSP%+/$
Return to the step in the main algorithm.

4.3.17 This section is referenced from the main flow of the algorithm and you should return to the step which called this one when done here. This section checks the sizes of elements within a character for correct width ratios. It uses the values of widest space and bar and bar pattern which were found previously. Now find the narrowest bar and space, and the sum of the nine elements following i (which make up the current character).

4.3.18 If the ratio of widest space to narrowest space is greater than max element ratio or less than min element ratio then return; the test failed.

4.3.19 If the ratio of widest bar to narrowest bar is greater than max element ratio then return; the test failed.

4.3.20 If bar pattern is greater than 0 and the ratio of widest bar to narrowest bar is less than min element ratio then return; the test failed.

4.3.21 If the ratio of narrowest bar t narrowest space is greater than max narrow element ratio then return; the test failed.

4.3.22 If the ratio of narrowest space to narrowest bar is greater than max narrow element ratio then return; the test failed.

4.3.23 Ok, return.

Section 5: Interleaved 2 of 5

5.1 General Description of Interleaved 2 of 5

See the referenced document for details. This is a binary code using elements of two widths to represent numeric characters. Each frame (10 elements) contains 2 characters, the first being represented by the bar pattern, the second by the space pattern. Each character pattern contains two wide elements and three narrow elements. The ratio of wide elements to narrow elements may vary over the range of 2:1 to 3:1 from one label to another, but is to be constant within a given label. There is no gap between adjacent characters. The actual scan direction is determined by the difference in the start and stop patterns at the beginning and ending of the label. The decoding process for each character treats the bars and spaces (and therefore each character) separately, in order to reduce errors due to systematic distortion of bar and space size. The number of data characters in a label is variable, but since characters are encoded in pairs, the number must be even. A fixed length is commonly used when decoding Interleaved 2 of 5, because of the relatively high probability that a partial scan of the label will yield seemingly valid data.

5.2 Overview of the algorithm

Variables and constants used in the decoding process are defined below, followed by a brief description of the decoding algorithm. A detailed description of the algorithm follows in the next section.

5.2.1 Status variables

Values of the status variables can be maintained globally to allow re-entrant use of the decoder.

Variable name	Description
i	pointer to the current element in the data buffer. This always points to a space when the decoder is called.
last char width	sum of the element widths in the last character pair decoded in the current label.
label string	decoded characters as an ASCII string.
forward	boolean; true if decoding in forward direction.
continue decode	boolean; true while the buffer at the current element looks like good data.
found label	boolean; true when a label has been put into label string.
data buffer	array of numbers representing widths of the elements scanned. They must alternate between bars and spaces.

5.2.2 Decode constants

Constant values are identified in this section. The constants are referenced by name in the description of the algorithm

Constant name	Value	Description
frame width	10	Number of elements in a character pair.
threshold ratio	.2188	Ratio of the wide/narrow decision point to the total width of the bars (spaces) in the character pair.
start stop threshold	1.5	Ratio of the second bar to the first bar of a start or stop pattern for determination of scan direction.
min element ratio	1.5	lower limit on the ratio of the width of the widest bar (space) to the narrowest bar (space) in a character.
max element ratio	5.0	upper limit on the ratio of the widest bar (space) to the narrowest bar (space) in a character pair.
max narrow element ratio	3.0	max ratio of the narrowest bar in a ratio character to the narrowest space, or vice versa.
max char ratio	1.25	max ratio of the sum of the elements in the current character pair to last char width.
min char ratio	.80	min ratio of the sum of the elements in the current character pair to last char width.
margin scaler	8	Value used to multiply starting bar and space elements in label by to make its width comparable to the width of the first (last) character pair.
max margin char ratio	2	max ratio of the sum of elements in the current character pair to the width of the start (stop) pattern scaled by margin scaler.
min margin ratio	3.0	min ratio of the width of the white space before the label to the sum of the width of the first two elements of the label.

5.2.3 Derivation of constants

threshold ratio

This is chosen to be near the midpoint of the difference between the ratio of a narrow element and a wide element to the total like element width in the character pair. For an N:1 wide/narrow ratio the value giving the midpoint is $t=((N+1)/2)/(3+2+N)$. For $N=3$ $t=.2222$; $N=2.5$ $t=.2188$; $N=2$ $t=.2143$. We are using .2188 (7/32). If an application only used a fixed ratio other than 2.5:1 this could be optimized to match.

start stop threshold

This threshold determines whether the pattern being examined is a start or stop pattern. A start pattern has two narrow bars, while a stop pattern has 1 narrow bar and 1 wide bar. This threshold is midway between the nominal narrow width (1) and the minimum wide/narrow ratio (2).

min element ratio

This is set to one half of the 2:1 minimum spec value for the wide/narrow ratio. Again, it and other limits could be changed for a particular application.

max element ratio

This is set to 5.0 based on the ratio of the widest wide element to the narrowest narrow element when the spec tolerances are applied to a nominal label. The actual worst case ratio is (3+.040+.0165)/(.040-.0165)=5.8.

max narrow element ratio

This is the only comparison other than total character width limiting the size of bars versus spaces. This is based on the spec ratio of a narrow element plus the tolerance to a narrow element minus the tolerance, plus some more to allow for scanning errors and printing errors beyond spec. The limit per spec is: (.040+.0165)/(.040-.0165)=2.40. Use 3.0. There is no theoretical upper limit to this, but it shouldn't be set to a value larger than will be seen in a label that is otherwise intact enough to decode.

max char ratio, min char ratio

These are chosen based on the possible scanning spot velocity variation within a character pair and the scanning and printing error over the character elements. Use 1.25 and .80 until better data for the particular scanning device being used is available.

min margin ratio

This is a compromise between enforcing the rigorous spec limits and program efficiency. The minimum white space per spec is 10 times the nominal element size. The sum of the first two elements of the label is always 2X. The largest element in the label can be 3X. Midpoint between 10 and 3 is 6.5X. The threshold ratio used is 3.0 giving a minimum margin of 6X.

5.2.4 General Decoding method

For efficiency the data buffer should be searched for a large white space indicating a potential label margin. Then all decoders can start processing from this point, avoiding duplicating the search process. If the decoder doesn't find a good label starting at this position it will exit back to the calling program. Before the decoder is first called some of the status variables must be initialized. They should be set as follows:

i : set to point to large white space.

Before any operation that looks at the data buffer, it is assumed that proper care will be taken to make sure that data is available in the buffer.

Each time the decoder is called it makes the comparison specified by min margin ratio. If this test passes, it looks for a forward or backward start pattern and tests the elements in the pattern using max narrow element ratio, min element ratio, and max element ratio. If this is all ok, the following variables are set:

continue decode : true
forward : true or false, depending on the pattern found
last char width : set to the sum of the elements in the start pattern * margin scaler
label string : set to empty
i : set to the current value of i + 4 (if forward is true, this points to the element before the first character pair in label. If forward is false, this points to last element of last character pair in label).

Then it continues going through the label elements, appending characters to the label string until an error occurs, the end pattern is found, or no data is available. For each character, tests using threshold ratio, min element ratio, max element ratio, and max narrow element ratio are applied plus the intercharacter checks for max char ratio, and min char ratio. If the pair is the first found, use the looser max margin char

ratio as the intercharacter check. If all tests passed, the status variables are updated:

last char width : set to the sum of the elements in the character pair just found
label string : the character pair found is appended to label string
i : set to the current value of i plus frame width

If any test fails, continue decode is set false.

If a character test failed, check to see if a stop pattern (or reverse start) pattern found, and valid trailing margin using min margin ratio, max element ratio, min element ratio, max like element ratio, max narrow element ratio, and max margin char ratio. If ok, do any secondary processing such as evaluating an optional check character. Then if everything is ok set found label true.

A check for a good character pattern is done by summing the widths of all the bars (spaces) in the character pair, multiplying the result by the threshold ratio, and using this value as the decision point for each bar (space) in the character pair. Note that this treats bars and spaces independently. The result is recorded as two binary patterns, one for bars and one for spaces, with ones indicating wide elements. The two binary numbers are used to find table entries indicating if a good character was read for each and what the character was. If a good character pattern for both characters in the pair was found, the smallest bar and space, and the total character width are determined. Tests for min and max element ratios are done independently for bars and spaces. The max narrow element ratio test limits the difference between the width of bars and spaces. If all these tests are ok, a good character pair was found.

5.3 Interleaved 2 of 5 Decode Algorithm

The decoding algorithm is given below. If any label integrity test fails, an exit from the algorithm will occur with the status variable continue decode set to false. Before calling the decoder set i to point to a possible margin (wide white space). Information in the scan data buffer is refered to as element(i) for the ith element of the data buffer. During the decoding process i is assumed to point to a margin, the space before the next character pair (forward decode), or the last space of the current character pair (backward decode). Follow the steps as specified, starting at 5.3.1.

5.3.1 Set found label false.

5.3.2 If less than frame width counts are available to be examined wait. (Check i + frame width against the last buffer location.)

5.3.3 If element(i) < min margin ratio * (element(i+1)+element(i+2)), then quit (margin too small).

5.3.4 Look for a valid start pattern:

Check if element(i+1)/element(i+2) > max narrow element ratio or element(i+2)/element(i+1) > max narrow element ratio. If so, quit. If not, determine direction of scan: Check if element(i+3)/element(i+1) > start stop threshold. If so, set forward to false, else set forward to true (start pattern has two narrow bars backward stop pattern has narrow bar followed by wide bar). If forward then check if element(i+2)/element(i+4) > min element ratio or element(i+4)/element(i+2) > min element ratio. If so, quit (two spaces in start pattern are not equal width). If forward also check if element(i+1)/element(i+3) > min element ratio. If so, quit (two bars are not the same width). If forward is false, the check if element(i+3)/element(i+1) < max element ratio. If not quit. Otherwise set continue decode true, set last char width to the sum of the elements i+1 and i+2 * margin scaler, set label string to empty, and increment i by 4. An apparent label start has been found.

5.3.5 If less than frame width counts are available to be examined (Check i + frame width against the last buffer location) wait.

5.3.6 Do the procedure specified starting at step 5.3.11 to get the character pattern. If a legal pattern wasn't found go to step 5.3.9 (no char found, check for start/stop pattern). Otherwise do the procedure starting at step 5.3.15 to check the element widths in the character. If they don't pass the tests, go to step 5.3.9 (character elements out of limits, check for start/stop pattern).

5.3.7 Compute the ratio of the sum of the elements in the current frame to last char width. If this is the first character pair (label strin is empty) then check if this ratio > max margin char ratio or (1/this ratio) > max margin char ratio. If it is, then quit (no characters found). If not first time thru, then if this ratio is greater than max char ratio or less than min char ratio go to step 5.3.9.

5.3.8 If the length of label string is the maximum allowable label length set continue decode false and quit (label string overflow). Otherwise set last char width to the width of the current frame, add frame width to i, and append the decoded character pair to label string. (If forward is true, then append bar character + space character to end of label string. If forward is false, append space character + bar character to the end of the label string.) Go to step 5.3.5.

5.3.9 Set continue decode to false.

5.3.10 Check for possible end of label:

If less than 4 counts available in the buffer then wait. (check i + 4 less than or equal to last buffer location). Otherwise, check if element(i+4)/(element(i+2)+element(i+3)) > min margin ratio. If not, quit. If so then check if element(i+3)/element(i+2) > max narrow element ratio or element(i+2)/element(i+3) > max narrow element ratio. If so, quit. If not then check that (element(i+2)+ element(i+3)) * margin scaler / last char width is greater than max margin char ratio or less than 1/max margin char ratio. If so, then quit. If not then if forward is true, then check that element(i+1)/element(i+3) is greater than max element ratio or less than min element ratio. If so, then quit. If forward is false then check that element(i+1)/element(i+3) is greater then start stop ratio or less than 1/min element ratio. If so, then quit. Also, if forward is false, check that element(i)/element(i+2) is greater than min element ratio or less than 1/min element ratio. If so then quit. Otherwise, if forward is false, then reverse order of label string. Then, do any optional operation for check sum. If no errors are found set found label true and quit.

5.3.11 This section is referenced from the main flow of the algorithm and should return to the step which called this one when done. If forward is true, elements i+1 to i+frame width will be summed (bars and spaces separately). If forward is false, then elements i to i+frame width-1 will be summed (bars and spaces separately). Find the widest bar, widest space, narrowest bar, and narrowest space in the corresponding 10 elements just examined. Multiply each of the sums (bar and space) by threshold ratio to find the wide/narrow breakpoint.

5.3.12 Set a binary number which will represent the bar pattern to 0. If forward is true, then use elements in order i+1, i+3, i+5, i+7, i+9. If forward is false, then use elements in order i+9, i+7, i+5, i+3, i+1. For each of the elements, multiply bar pattern by 2, and increment it by 1 if the element under consideration is greater than the bar threshold calculated in step 5.3.11.

5.3.13 Set a binary number which will represent the space pattern to 0. If forward is true, then use elements in order i+2, i+4 i+6, i+8, i+10. If forward is false, then use elements in order i+8, i+6, i+4, i+2, i. For each of the elements, multiply space pattern by 2, and increment it by 1 if the element under consideration is greater than the space threshold calculated in step 5.3.11.

5.3.14 Use the binary number generated by the bar pattern to as a pointer to select a character from the following list, indexed at 0. for example, if the pattern had a value of 2, select the third element in the list. Repeat the above to select the space character using the space pattern.

Character list:
XXX7X40XX29X6XXXX18X5XXX3XXXXXXX

If the character selected for either bar or space pattern is X then indicate bad pattern to main algorithm, otherwise indicate good pattern.
Return to the step in the main algorithm.

5.3.15 This section is referenced from the main flow of the algorithm and should return to the step which called this one when done. This section checks the sizes of elements within a character for correct width ratios. It uses the values of widest and narrowest space and bar and bar pattern which were found previously. Add bar pattern width to space pattern width to get total char width.

5.3.16 If the ratio of widest space to narrowest space is greater than max element ratio or less than min element ratio then return; the test failed.

5.3.17 If the ratio of widest bar to narrowest bar is greater then max element ratio then return; the test failed.

5.3.18 If the ratio of widest bar to narrowest bar is less than min element ratio then return; the test failed.

5.3.19 If the ratio of narrowest bar to narrowest space is greater than max narrow element ratio then return; the test failed.

5.3.20 If the ratio of narrowest space to narrowest bar is greater than max narrow element ratio then return; the test failed.

5.3.21 Ok, return.

Section 6: Codabar

6.1 General Description of Codabar

See the referenced document for details. This is a binary code using elements of two widths to represent the ten digits and the six characters -$:/.+ plus four start/stop characters denoted A,B,C,D. The 20 character patterns each contain 7 elements. The wide/narrow ratio may range from 2:1 to 3:1 from label to label, but is to be constant in a given label. Twelve of the characters contain 2 wide elements while the other 8 contain 3 wide elements. This results is two possible nominal character widths. Characters are seperated by an intercharacter gap. A variable number of characters are allowed, but 32 is a common upper limit. The start/stop characters are considered part of the label information, and in some implementations control concatenation. An optional check character is also defined.

6.2 Overview of the algorithm

Variables and constants used in the decoding process are defined below, followed by a brief description of the decoding algorithm. A detailed description of the algorithm follows in the next section.

6.2.1 Status variables

Values of the status variables can be maintained globally to allow re-entrant use of the decoder.

Variable name	Description
i	pointer to the current element in the data buffer. This always points to a space when the decoder is called.
last char width	sum of the element widths in the last character decoded in the current label. This does't include the intercharacter gap.
label string	decoded characters as an ASCII string.
forward	boolean; true if decoding in forward direction.
continue decode	boolean; true while the buffer at the current element looks like good data.
found label	boolean; true when a label has been put into label string.
data buffer	array of numbers representing widths of the elements scanned. They must alternate between bars and spaces.

6.2.2 Decode constants

Constant values are identified in this section. The constants are referenced by name in the description of the algorithm

Constant name	Value	Description
frame width	8	Number of elements in a character plus the gap.
threshold ratio	.70	Ratio of the width of the widest bar (space) in a character to the wide/narrow decision point.
min element ratio	1.5	lower limit on the ratio of the width of the widest bar (space) to the narrowest bar (space) in a character.
max element ratio	5.0	upper limit on the ratio of the width of the widest bar (space) to the narrowest bar (space) in a character.
max narrow element ratio	3.0	max ratio of the narrowest bar in a character to the narrowest space, or vice versa.
max char ratio	1.25	max ratio of the sum of the elements in the current character to last char width, not including the intercharacter gap.
min char ratio	.80	min ratio of the sum of the elements in the current character to last char width, not including the intercharacter gap.
max gap ratio	30	max ratio of the sum of the elements in the current character to the previous intercharacter gap.
min gap ratio	1.5	min ratio of the sum of the elements in the current character to the previous intercharacter gap.
min margin ratio	1.0	min ratio of the width of the white space before the label to the sum of the width of the first three elements of the label.

6.2.3 Derivation of constants

threshold ratio

This is chosen to be near the midpoint of the difference between a narrow element and a wide element. For a N:1 wide/narrow ratio the value giving the midpoint is t=((N-1)/2)+1)*N. For N=3 t=.6666; N=2.5 t=.7000; N=2 t=.7500. We are using .7000. If a application only used a fixed ratio other than 2.5:1 this could be optimized to match. If labels are rejected due to a wide variation in the width of the narrow elements in the characters s/+% (because of the test for all narrow elements) the ratio could be decreased.

min element ratio

This is set to one half of the 2:1 minimum spec value for the wide/narrow ratio. Again, it and other limits could be changed for a particular application.

max element ratio

This is set to 5.0 based on the ratio of the widest wide element to the narrowest narrow element when the spec tolerances are applied to a nominal label. The actual worst case ratio is (3*.040+.009)/(.040-.009)=4.16.

max narrow element ratio

This is the only comparison other than total character width limiting the size of bars versus spaces. This is based on the spec ratio of a narrow element plus the tolerance to a narrow element minus the tolerance, plus some more to allow for scanning errors and printing errors beyond spec. The limit per spec is (.040+.009)/(.040-.009)=1.58. Use 3.0. There is no theoretical upper limit to this, but it shouldn't be set to a value larger than will be seen in a label that is otherwise intact enough to decode.

max char ratio, min char ratio

These are chosen based on the possible scanning spot velocity variation within a character and the scanning and printing error over the character elements. Character width may vary because of the data containing characters with 2 wide elements or 3 wide elements. At a 3:1 wide/narrow ratio this can result in characters containing 11 or 13 nominal elements. This causes a variation of 11/13 or 13/11 in character width before taking into account other sources of difference. Use 1.25 and .80 until better data for the particular scanning device being used is available.

max gap ratio

This is based on the ratio of the sum of the width of the elements in the widest legal character (excluding the gap) to the narrowest gap. In a label with a 3:1 wide/narrow ratio the widest character is 13 nominal elements wide. At .040 inch nominal element size the minimum gap is .040-.009 or .78 nominal. This gives a ratio of 13/.78 = 17. Use 30 to allow for out of spec gap widths.

min gap ratio

This is based on the ratio of the sum the width of the of the elements the narrowest legal character (excluding the gap) to the widest gap. In a label with a 2:1 wide/narrow ratio a narrow character is 9 nominal elements wide. The widest gap is 5.3 times a nominal element per spec, for a ratio of 9/5.3 = 1.7. Use 1.5 to allow some extra margin for error.

min margin ratio

This is a compromise between enforcing the rigorous spec limits and program efficiency. The minimum white space per spec is 10 times the nominal element size. The sum of the first three elements of the label for a forward label can be 3 to 5 times nominal; for a backward label it can be 4 to 7 times nominal. This results is a min margin of 3 to 7 nominal elements, which allows for out of spec labels and scanning error.

6.2.4 General Decoding method

For efficiency the data buffer should be searched for a large white space indicating a potential label margin. Then all decoders can start processing from this point, avoiding duplicating the search process. If the decoder doesn't find a good label starting at this position it will exit back to the calling program. Before the decoder is first called some of the status variables must be initialized. They should be set as follows:

i : set to point to large white space.

Before any operation that looks at the data buffer, it is assumed that proper care will be taken to make sure that data is available in the buffer.

Each time the decoder is called it makes the comparison specified by min margin ratio. If this test passes, it looks for a forward or backward start character pattern (A,B,C or D) and tests the elements in the character using threshold ratio, min element ratio, max element ratio, and max narrow element ratio. If this is all ok, the following variables are set:

continue decode : true
forward : true or false, depending on the character found
last char width : set to the sum of the elements in the start character
label string : set to empty
i : set to the current value of i + frame width.

Then it continues going through the label elements, appending characters to the label string until an error occurs, the end (A,B,C or D) character is found, or no data is available. For each character, the same

checks made for a start character are applied (except the min margin ratio) plus the intercharacter checks for max char ratio, min char ratio, max gap ratio and min gap ratio. If the character found wasn't a stop character and all tests passed, the status variables are updated:

last char width : set to the sum of the elements in the character just found
label string : the character found is appended to label string
i : set to the current value of i plus frame width

If any test fails, continue decode is set false.

If the character was a stop character, set continue decode false, and check for the trailing margin using min margin ratio. If ok, do any secondary processing such as reversing the label string to correct for a backward scan, evaluating an optional check character, label concatenation, etc. Then if everything is ok set found label true.

The check for a good character pattern is done by finding the widest bar and space in the character, multiplying each by the threshold ratio, then comparing the result to each bar and space in the character to identify the wide and narrow elements. Note that this treats bars and spaces independently. The result is recorded as a binary pattern, one bit per bar or space, with ones indicating wide elements. Since it is possible for a Codabar character pattern to have no wide spaces (which would look like all wide spaces to the above procedure), a separate test must be performed to detect and correct the binary pattern. The binary number resulting from this process is used to select a table entry indicating if a good character was read and what the character was. If a good character pattern was found, the smallest bar and space, and the total character width are determined. Tests for min and max element ratios are done independently for bars and spaces, taking into account the case of all narrow bars being found. The max narrow element ratio test limits the difference between the width of bars and spaces. If all these tests are ok, a good character was found.

6.3 Codabar Decode Algorithm

The decoding algorithm is given below. If any label integrity test fails, an exit from the algorithm will occur with the status variable continue decode set to false. Before calling the decoder set i to a possible margin (wide white space). Information in the scan data buffer is refered to as element(i) for ith element of the data buffer. During the decoding process i is assumed to point to a margin or intercharacter gap. Follow the steps as specified, starting at 6.3.1.

6.3.1 Set found label false.

6.3.2 If less then frame width counts are available to be examined wait. (Check i + frame width against the last buffer location.)

6.3.3 If min margin ratio > (element(i) / (the sum of element(i+1) through element(i+3)) quit (margin too small).

6.3.4 Use the procedure at starting at step 6.3.11 to check for a character. If forward character is A, B, C, or D set forward true. If backward character is A, B, C, or D set forward false. If neither was found quit (no start char found). Otherwise do the procedure specified starting at step 6.3.16 to check the element widths in the character. If they don't pass the tests, quit (character elements out of limits). Otherwise set continue decode true, set last char width to the sum of the elements in the character computed at step 6.3.16, set label string to emtpy, and increment i by frame width. An apparent label start has been found.

6.3.5 If less then frame width counts are available to be examined wait. (Check i + frame width against the last buffer location.

6.3.6 Do the procedure specified starting at step 6.3.11 to get the character pattern. If a legal pattern wasn't found set continue decode false and quit (no char found). Otherwise do the procedure starting at step 6.3. 16 to check the element widths in the character. If they don't pass the tests, set continue decode false and quit (character elements out of limits).

6.3.7 Compute the ratio of the sum of the elements in the current character to last char width. If this ratio is greater than max char ratio or less than min char ratio set continue decode false and quit.

6.3.8 Compute the ratio of the sum of the elements in the current character to element(i). If this ratio is greater than max gap ratio or less than min gap ratio set continue decode false and quit.

6.3.9 If forward is true then the current charcter is the forward character found in step 6.3.8, otherwise the current character is the backward character. If the current character is A, B, C, or D go to 6.3.10. Otherwise if the length of label string is the maximum allowable label length set continue decode false and quit (label string overflow). Otherwise set last char width to the width of the current character, add frame width to i, and append the current character to label string. Go to step 6.3.5.

6.3.10 Set continue decode false. If min margin ratio > element(i+framewidth) / (the sum of element-(i+5) through element(i+7)) then quit. Otherwise if forward is false reverse label string, and do any optional operations for check sum or concatenation (per the referenced spec). If no errors are found set found label true and quit.

6.3.11 This section is referenced from the main flow of the algorithm and you should return to the step which called this one when done here. The elements of the current character will be examined looking for a good character pattern. Find the widest bar and widest space in the seven elements i+1 through i+7. Multiply each of these by threshold ratio to find the wide/narrow breakpoint.

6.3.12 Set a binary number which will represent the wide/narrow pattern to 0. For each of the elements i+1, through i+7 multiply pattern by 2, and increment it by 1 if the element under consideration is greater than the bar or space threshold calculated in step 4.3.11 (use the bar or space threshold as appropriate).

6.3.13 If no narrow space was found set pattern to pattern and 1010101. This will correct for the case of no wide spaces in a character looking appearing to be all wide spaces because of the method used.

6.3.14 Now determine if pattern is one of the allowable values. One way to do this is to use a table having 128 entries to check for a valid pattern and convert it to an index to a character string. If the entry in pattern table (see below) is 0 for the pattern found, the test failed; return. Otherwise set char pointer to the value found in pattern table.

```
pattern_table    :array[0..127]  =
                 {             0   1   2   3   4   5   6   7   8   9 }

                 { 0}     (    0,  0,  0,  1,  0,  0,  2,  0,  0,  3,
                 {10}          0,  4,  5,  0,  6,  0,  0,  0,  7,  0,
                 {20}          0,  8,  0,  0,  9,  0, 10,  0,  0,  0,
                 {30}          0,  0,  0, 11,  0,  0, 12,  0,  0,  0,
                 {40}          0, 13,  0,  0, 14,  0,  0,  0, 15,  0,
                 {50}          0,  0,  0,  0,  0,  0, 16,  0,  0,  0,
                 {60}          0,  0,  0,  0,  0,  0, 17,  0,  0, 18,
                 {70}          0,  0, 19,  0, 20,  0,  0,  0,  0,  0,
                 {80}          0, 21,  0,  0, 22,  0,  0,  0,  0,  0,
                 {90}          0,  0,  0,  0,  0,  0, 23,  0,  0,  0,
                 {100}         0,  0,  0,  0, 24,  0,  0,  0,  0,  0,
                 {110}         0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
                 {120}         0,  0,  0,  0,  0,  0,  0,  0);
```

6.3.15 Use char pointer to select forward character and backward character from these two lists of 24 characters. For example, if char pointer is two, pick the second character from each list, and so on.
Forward characters: 012C-D4+$A467BX8X5:9X/.3X
Backward characters: 389X$X7.-X54XA1D6/2B:+0C
Return to the step in the main algorithm.

6.3.16 This section is referenced from the main flow of the algorithm and you should return to the step which called this one when done here. This section checks the sizes of elements within a character for correct width ratios. It uses the values of widest space and bar and bar pattern which were found previously. Now find the narrowest bar and space, and the sum of the seven elements following i (which make up the current character).

6.3.17 If the ratio of widest bar to narrowest bar is greater than max element ratio or less than min element ratio then return; the test failed.

6.3.18 If the ratio of widest space to narrowest space is greater than max element ratio then return; the test failed.

6.3.19 If wide spaces were found and the ratio of widest space to narrowest space is less than min element ratio then return; the test failed.

6.3.20 If the ratio of narrowest bar to narrowest space is greater than max narrow element ratio then return; the test failed.

6.3.21 If the ratio of narrowest space to narrowest bar is greater than max narrow element ratio then return; the test failed.

6.3.22 Ok, return.

Section 7: Code 93

7.1 General Description of Code 93

See the referenced document for details. This is a codė using characters made up of six elements per character with four different width elements used. The narrowest element is defined as being one module wide, with the others being two, three and four modules wide. A character is nine modules wide. There are 48 character patterns defined, with some being used as control characters in a system for encoding the full ASCII character set. Each label contains a variable number of data characters and two mandatory check characters. A method for concatenating labels is also defined. The scan direction is recognized by looking for a forward or backward start character. The code is designed for like edge to like edge decoding. Two term sums are computed and used to decode the characters.

7.2 Overview of the Algorithm

Variables and constants used in the decoding process are defined below, followed by a brief description of the decoding algorithm. A detailed description of the algorithm follows in the next section.

7.2.1 Status variables

Values of the status variables can be maintained globally to allow re-entrant use of the decoder.

Variable name	Description
i	pointer to the current element in the data buffer. This always points to a space when the decoder is called.
last char width	sum of the element widths in the last character decoded in the current label.
label string	decoded characters as an ASCII string.
forward	boolean; true if decoding in forward direction.
continue decode	boolean; true while the buffer at the current element looks like good data.
found label	boolean; true when a label has been put into label string.
data buffer	array of numbers representing widths of the elements scanned. They must alternate between bars and spaces.

7.2.2 Decode constants

Constant values are identified in this section. The constants are referenced by name in the description of the algorithm.

Constant name	Value	Description
frame width	6	Number of elements in a character.
threshold ratio 1	2.5/9	Used to determine value of two term sums.
threshold ratio 2	3.5/9	
threshold ratio 3	4.5/9	
max element ratio	8.0	upper limit on the ratio of the width of the widest bar (space) to the narrowest bar (space) in a character.
max char ratio	1.25	max ratio of the sum of the elements in the current character to last char width, not including the intercharacter gap.
min char ratio	.80	min ratio of the sum of the elements in the current character to last char width, not including the intercharacter gap.
min margin ratio	1.0	min ratio of the width of the white space before the label to three times the sum of the width of the first two elements of the label .
stop element ratio	2.0	max value of either the ratio of the width of the last bar of a stop character to the next to the last bar, or the inverse of that ratio.

7.2.3 Derivation of constants

threshold ratio 1

This is chosen to be near the midpoint of the difference between a two term sum of two one module elements (2 modules total) and a two term sum of a one module element and a two module element(3 modules total). It is normalized to the 9 module character size.

threshold ratio 2

Similar to threshold 1, it is the breakpoint between 3 module and 4 module two term sums.

threshold ratio 3

This is the breakpoint between 4 module and 5 module two term sums.

max char ratio, min char ratio

These are chosen based on the possible scanning spot velocity variation within a character and the scanning and printing error over the character elements. Use 1.25 and 0.80 until better data for the particular scanning device being used is available.

min margin ratio

This is a compromise between enforcing the rigorous spec limits and program efficiency. The minimum white space per spec is 10 times the nominal module size. The sum of the first two elements of a forward or backward label is 2 nominal modules. Checking for a margin of at least three times this results is a min margin of 6 nominal modules, which allows for out of spec labels and scanning error.

stop element ratio

This is not critical, and is used to check that the width of the last bar is similar to the width of the other elements in the stop character.

7.2.4 General Decoding method

For efficiency the data buffer should be searched for a large white space indicating a potential label margin. Then all decoders can start processing from this point, avoiding duplicating the search process. If the decoder doesn't find a good label starting at this position it will exit back to the calling program. Before the decoder is first called some of the status variables must be initialized. They should be set as follows:

i : set to point to large white space.

Before any operation that looks at the data buffer, it is assumed that proper care will be taken to make sure that data is available in the buffer.

Each time the decoder is called it makes the comparison specified by min margin ratio. If this test passes, it looks for a forward or backward start character pattern and tests the elements in the character using the threshold ratios and max element ratio. If this is all ok, the following variables are set:

continue decode : true
forward : true or false, depending on the character found
last char width : set to the sum of the elements in the start character
label string : set to empty
i : set to the current value of i + frame width.

Then it continues going through the label elements, appending characters to the label string until an error occurs, the end character is found, or no data is available. For each character, the same checks made for a start character are applied (except the min margin ratio) plus the intercharacter checks for max char ratio and min char ratio. If the character found wasn't a stop character and all tests passed, the status variables are updated:

last char width : set to the sum of the elements in the character just found
label string : the character found is appended to label string
i : set to the current value of i plus frame width

If any test fails, continue decode is set false.

If the character was a stop character, set continue decode false, and check for the trailing margin using min margin ratio. If ok, do any secondary processing such as reversing the label string to correct for a backward scan, evaluating the two check characters, optional label concatenation, etc. Then if everything is ok set found label true.

The check for a good character pattern is done by finding the sum of the elements in the character

(total width), then computing three threshold values by multiplying each of the three threshold ratio values by the total width. Then four two term sums are calculated, and a determination made of whether each sum is 2,3,4, or 5 modules in size. The resulting four digits are used to look up the proper character code. If the label is being scanned backward the two term sums are calculated from the other end of the character, resulting in the same set of sums. Provision must be made to find a forward or backward start character in order to identify direction initially, but all other characters will always have a single representation. If a good character is found the max element ratio is checked. This requires finding the narrowest and widest bars and spaces. If this test is ok a the character is ok.

7.3 Code 93 Decode Algorithm

The decoding algorithm is given below. If any label integrity test fails, an exit from the algorithm will occur with the status variable continue decode set to false. Before calling the decoder set i to a possible margin (vide white space). Information in the scan data buffer is refered to as element(i) for the ith element of the data buffer. During the decoding process i is assumed to point to a margin or space. Follow the steps as specified, starting at 7.3.1.

7.3.1 Set found label false.

7.3.2 If less then frame width counts are available to be examined wait. (Check i + frame width against the last buffer location.)

7.3.3 If min margin ratio > (element(i) / 3*(the sum of element(i+1) and element(i+2)) quit (margin too small).

7.3.4 Set forward true. Use the procedure at starting at step 7.3.12 determine the hex value representing the character. If pattern is 2225 the label really is forward. If pattern is 2552 then set forward false. If neither was found quit (no start char found). If forward is false and element(i+1)/element(i+3) > stop element ratio, or element(i+3)/element(i+1) > stop element ratio, quit. Otherwise do the procedure specified starting at step 7.3.17 to check the element widths in the character. If they don't pass the tests, quit (character elements out of limits). Otherwise set continue decode true, set last char width to the sum of the elements in the character computed at step 7.3.12, set label string to empty, and increment i by frame width. An apparent label start has been found.

7.3.5 If less then 8 counts are available to be examined wait. (Check i + frame width against the last buffer location.)

7.3.6 Do the procedure specified starting at step 7.3.12 to get the character pattern. Look up the character corresponding to the pattern found. A fast method should be used. A decision tree which branches at each digit for the first one or two digits and has the character values at the leaves can be used. The following table gives the pattern to character conversion. The start and special character patterns are denoted by otherwise unused characters. (The characters actually sent from the decoder are chosen in a later step).

```
2222: char = '7'
2223: char = 'L'
2225: char = '(' fwd start
2233: char = '1'
2234: char = 'M'
2244: char = '2'
2245: char = 'N'
2255: char = '3'
2332: char = 'G'
2333: char = 'W'
2334: char = '/'
2343: char = 'H'
2344: char = 'X'
2354: char = 'I'
2443: char = '+'
2552: char = ')' bkwd start
3222: char = 'A'
3223: char = 'S'
3224: char = '%'
3233: char = 'B'
3234: char = 'T'
3244: char = 'C'
3322: char = '4'
3323: char = 'O'
3324: char = '-'
3332: char = 'Q'
3333: char = '5'
3334: char = '!' ^$ char
3343: char = 'R'
3344: char = '6'
3432: char = 'J'
3433: char = 'Y'
3443: char = '&' ^+ char
3542: char = 'Z'
4222: char = '.'
4223: char = '#' ^/ char
4233: char = ' '
4322: char = 'D'
4323: char = 'U'
4332: char = '@' ^% char
4333: char = 'E'
4422: char = '0'
4423: char = 'P'
4432: char = 'V'
4433: char = '8'
4532: char = 'K'
5322: char = '$'
5422: char = 'F'
5522: char = '9'
```

7.3.7 If a legal pattern wasn't found set continue decode false and quit (no char found). Otherwise do the procedure starting at step 7.3.17 to check the element widths in the character. If they don't pass the tests, set continue decode false and quit (character elements out of limits).

7.3.8 Compute the ratio of the sum of the elements in the current character to last char width. If this ratio is greater than max char ratio or less than min char ratio set continue decode false and quit.

7.3.9 If current character is start/stop then go to 7.3.11.

7.3.10 If the length of label string is the maximum allowable label length set continue decode false and quit (label string overflow). Otherwise set last char width to the width of the current character, add frame width to i, and append the current character to label string. Go to step 7.3.5.

7.3.11 Set continue decode false. If min margin ratio > element(i+8) / 3*(the sum of element(i+6) plus element(i+7)) then quit. If forward is true then quit if element(i+7)/element(i+5) > stop element ratio, or if element(i+5)/element(i+7) > stop element ratio. Otherwise if forward is false reverse label string, and do the check sum calculations. If ok search through the label for any shift characters and replace the shift character and the following character with the correct character according to the table in the referenced spec. The shift characters are represented in the label string as follows:

circle $ is a !

circle + is a &

circle % is an @

circle / is a # .

Next do any optional operations such as concatenation (per the referenced spec). If no errors are found set found label true.

Quite.

7.3.12 This section is referenced from the main flow of the algorithm and you should return to the step which called this one when done here. The elements of the current character will be examined looking for a good character pattern. If forward is true find the total character width by adding element(i+1) through element(i+6), otherwise by adding element(i+2) through element(i+7). Now if forward is true go to step 7.3.13, otherwise go 7.3.14.

7.3.13 Calculate four two term sums T1 through T4 as follows.

T1 = element(i+1) + element(i+2)

T2 = element(i+2) + element(i+3)

T3 = element(i+3) + element(i+4)
T4 = element(i+4) + element(i+5)
Go to 7.3.15.

7.3.14 Calculate four two term sums T1 through T4 as follows.
T1 = element(i+6) + element(i+7)
T2 = element(i+5) + element(i+6)
T3 = element(i+4) + element(i+5)
T4 = element(i+3) + element(i+4)

7.3.15 Compute three threshold values thresh 1, thresh 2, thresh 3 by multiplying the total character width times threshold ratio 1 through threshold ratio 3.

7.3.16 Compute the four digits D1 through D4 of the pattern by doing the following for each sum T1 through T4. For j 1 through 4 do the following:
Dj = 2 if Tj < thresh 1.
Dj = 3 if thresh 1 <= Tj < thresh 2.
Dj = 4 if thresh 2 <= Tj < thresh 3.
Dj = 5 if thresh 3 <= Tj.
Then pattern is equal to D4 + 16*D3 + 256*D2 + 4096*D1 (do this by shifting and adding as they are computed). Return to the step in the main algorithm.

7.3.17 This section is referenced from the main flow of the algorithm and you should return to the step which called this one when done here. This section checks the sizes of elements within a character for correct width ratios. Find the widest and narrowest bar and space.

7.3.18 If the ratio of widest bar to narrowest bar is greater than max element ratio then return; the test failed.

7.3.19 If the ratio of widest space to narrowest space is greater than max element ratio then return; the test failed.

7.3.20. Ok, return.

Section 8: Code 128

8.1 General Description of Code 128

See the referenced document for details. This is a code using characters made up of six elements per character with four different width elements used. The narrowest element is defined as being one module wide, with the others being two, three and four modules wide. A character is eleven modules wide. The bars in a character are made up of an even number of modules (even parity). Character parity is checked in the decoding process. There are 107 character patterns defined, including three different start characters, one stop character, several to select which of the three different pattern to character translations to use (code A, B, or C), and four decoder cc functions. The choice of start character determines which pattern to character translation is used initially. Each label contains a variable number of data characters and a mandatory check character. A method for concatenating labels is also defined. The scan direction is recognized by looking for a forward or backward start or stop character. The code is designed for like edge to like edge decoding. Two term sums are computed and used to decode the characters.

8.2 Overview of the Algorithm

Variables and constants used in the decoding process are defined below, followed by a brief description of the decoding algorithm. A detailed description of the algorithm follows in the next section.

8.2.1 Status variables

Values of the status variables can be maintained globally to allow re-entrant use of the decoder.

Variable name	Description
i	pointer to the current element in the data buffer. This always points to a space when the decoder is called.
last char width	sum of the element widths in the last character decoded in the current label.
label string	decoded characters as an ASCII string.
forward	boolean; true if decoding in forward direction.
continue decode	boolean; true while the buffer at the current element looks like good data.
found label	boolean; true when a label has been put into label string.
data buffer	array of numbers representing widths of the elements scanned. They must alternate between bars and spaces.

8.2.2 Decode constants

Constant values are identified in this section. The constants are referenced by name in the description of the algorithm

Constant name	Value	Description
frame width	6	Number of elements in a character.
threshold ratio 1	2.5/11	Used to determine value of two term sums.
threshold ratio 2	3.5/11	
threshold ratio 3	4.5/11	
threshold ratio 4	5.5/11	
threshold ratio 5	6.5/11	
max element ratio	8.0	upper limit on the ratio of the width of the widest bar (space) to the narrowest bar (space) in a character.
max char ratio	1.25	max ratio of the sum of the elements in the current character to last char width, not including the intercharacter gap.
min char ratio	.80	min ratio of the sum of the elements in the current character to last char width, not including the intercharacter gap.
min margin ratio	1.0	min ratio of the width of the white space before the label to two times the sum of the width of the first two elements of the label.
stop element ratio	2.0	max value of either the ratio of the last bar of a stop char to the first bar, or the inverse of that ratio.

8.2.3 Derivation of constants

threshold ratio 1

This is chosen to be near the midpoint of the difference between a two term sum of two one module elements (2 modules total) and a two term sum of a one module element and a two module element(3 modules total). It is normalized to the 11 module character size.

threshold ratio 2

Similar to threshold 1, it is the breakpoint between 3 module and 4 module two term sums.

threshold ratio 3

This is the breakpoint between 4 module and 5 module two term sums.

threshold ratio 4

This is the breakpoint between 5 module and 6 module two term sums.

threshold ratio 5

This is the breakpoint between 6 module and 7 module two term sums.

max char ratio, min char ratio

These are chosen based on the possible scanning spot velocity variation within a character and the scanning and printing error over the character elements. Use 1.25 and .80 until better data for the particular scanning device being used is available.

min margin ratio

This is a compromise between enforcing the rigerous spec limits and program efficiency. The minimum white space per spec is 10 times the nominal module size. The sum of the first two elements of a forward or backward label is 3 nominal modules. Checking for a margin of at least two times this results is a min margin of 6 nominal modules, which allows for out of spec labels and scanning error.

stop element ratio

This is not critical, and is used to check that the width of the last bar is similar to the width of the other elements in the stop character.

8.2.4 General decoding method

For efficiency the data buffer should be searched for a large white space indicating a potential label margin. Then all decoders can start processing from this point, avoiding duplicating the search process. If the decoder doesn't find a good label starting at this position it will exit back to the calling program. Before the decoder is first called some of the status variables must be initialized. They should be set as follows:

i : set to point to large white space.

Before any operation that looks at the data buffer, it is assumed that proper care will be taken to make

sure that data is available in the buffer.

Each time the decoder is called it makes the comparison specified by min margin ratio. If this test passes, it looks for a forward or backward start or stop character pattern and tests the elements in the character using the threshold ratios and max element ratio. If this is all ok, the following variable are set:

continue decode : true
forward : true or false, depending on the character found
last char width : set to the sum of the elements in the start character
label string : set to empty
i : set to the current value of i + frame width.

All processing to translate code sets A, B, and C takes place after the complete label has been found.

Next it continues going through the label elements, appending characters to the label string until an error occurs, the end character is found, or no data is available. For each character, the same checks made for a start character are applied (except the min margin ratio) plus the intercharacter checks for ratio and min char ratio. If the character found wasn't a stop or backward start character and all tests passed, the status variables are updated.

last char width : set to the sum of the elements in the character just found
label string : the character found is appended to label string
i : set to the current value of i plus frame width

If any test fails, continue decode is set false.

If the character was a stop or backward starter, set continue decode false, and check for the trailing margin using min margin ratio. If ok, do any secondary processing such as reversing the label string to correct for a backward scan, evaluating the two check characters, optional label concatenation, etc. Then if everything is ok set found label true.

The check for a good character pattern is done by finding the sum of the elements in the character (total width), then computing five threshold values by multiplying each of the five threshold ratio values by the total width. Then four two term sums are calculated, and a determination made of whether each sum is 2,3,4,5,6 or 7 modules in size. The resulting four digits are used to look up the proper character code. If the label is being scanned backward the two term sums are calculated from the other end of the character, resulting in the same set of sums. Provision must be made to find forward start characters and forward or backward stop characters in order to identify direction initially, but all other characters will always have a single representation. If a good character is found the max element ratio is checked. This requires finding the narrowest and widest bars and spaces. If this test is ok a the character is ok.

8.3 Code 128 Decode Algorithm

The decoding algorithm is given below. If any label integrity test fails, an exit from the algorithm will occur with the status variable continue decode set to false. Before calling the decoder set I to a possible margin (wide white space). Information in the scan data buffer is refered to as element(I) for the Ith element of the data buffer. During the decoding process i is assumed to point to a margin or space. Follow the steps as specified, starting at 8.3.1.

8.3.1 Set found label false.

8.3.2 If less then frame width counts are available to be examined wait. (Check I + frame width against the last buffer location.)

8.3.3 if min margin ratio > (element(i) / 2*(the sum of element(i+1) and element(i+2)) quit (margin too small).

8.3.4 Set forward true. Use the procedure at starting at step 8.3.11 to determine the hex value representing the character. If pattern is any of the following set label string as shown:

```
pattern: 3255  label string: character 103 (start A)
         3233                          104 (start B)
         3235                          105 (start C)
         3224                          107 (backward stop)
```

If pattern was backward stop set forward false. If none of the four patterns were found quit. Check the

character parity as specified in step 8.3.7. If bad quit (parity error). Otherwise do the procedure specified starting at step 8.3.16 to check the element widths in the character. If they don't pass the tests, quit (character elements out of limits). Otherwise if forward false and element(i+7)/element(i+1) > stop element ratio, or if forward false and element(i+1)/element(i+7) > stop element ratio, quit. Otherwise set continue decode true, set last char width to the sum of the elements in the character computed at step 8.3.11, and increment i by frame width. An apparent label start has been found.

8.3.5 If less then 8 counts are available to be examined wait. (Check i + frame width against the last buffer location.)

8.3.6 Do the procedure specified starting at step 8.3.11 to get the character pattern. Look up the character corresponding to the pattern found. A fast method should be used. A decision tree which branches at each digit for the first one or two digits and has the character values at the leaves can be used. The following table gives the pattern to character conversion. The start and special character patterns are denoted by otherwise unused characters. (The characters actually sent from the decoder are chosen in a later step.

2225 char= 92	3434 char= 13	4534 char= 44
2234 char= 63	3442 char= 51	4542 char= 22
2236 char= 80	3443 char= 6	4543 char= 8
2245 char= 33	3444 char= 53	4552 char= 60
2247 char= 93	3445 char= 14	4553 char= 18
2256 char= 64	3453 char= 21	4554 char= 38
2334 char= 42	3454 char= 7	4643 char= 47
2343 char= 69	3464 char= 52	4752 char= 79
2345 char= 12	3465 char= 72	5222 char= 97
2354 char= 36		5224 char= 102
2356 char= 43	3543 char= 16	5233 char= 86
2365 char= 70	3553 char= 90	5244 char= 98
2443 char= 45	3554 char= 17	5323 char= 25
2445 char= 99	3652 char= 84	5333 char= 91
2454 char= 15	3663 char= 85	5334 char= 26
2465 char= 46	4223 char= 54	5422 char= 40
2552 char= 95	4225 char= 101	5424 char= 50
2554 char= 100	4234 char= 24	5432 char= 28
2563 char= 83	4245 char= 55	5433 char= 11
2574 char= 96	4322 char= 76	5442 char= 77
3224 char= 107 bkwd stop	4324 char= 19	5443 char= 29
3233 char= 104 start b	4332 char= 57	5444 char= 41
3235 char= 105 start c	4333 char= 9	5523 char= 67
3244 char= 39	4334 char= 23	5533 char= 32
3246 char= 49	4335 char= 20	5534 char= 68
3255 char= 103 start a	4343 char= 27	5632 char= 73
3323 char= 65	4344 char= 10	5642 char= 106 fwd stop
3325 char= 81	4354 char= 58	5643 char= 74
3333 char= 30	4355 char= 61	6322 char= 87
3334 char= 3	4423 char= 34	6333 char= 88
3335 char= 89	4425 char= 94	6423 char= 56
3336 char= 82	4433 char= 1	6522 char= 78
3344 char= 0	4434 char= 5	6532 char= 59
3345 char= 4	4443 char= 48	6533 char= 75
3355 char= 31	4444 char= 2	7422 char= 62
3356 char= 66	4445 char= 35	
3432 char= 71	4532 char= 37	

8.3.7 If a legal pattern wasn't found set continue decode false and quit (no char found). Otherwise check for a parity error by using the character value just found as an index into the following table to get a value V. Then if (V+1.75)/11 < bar total or (V-1.75)/11>bar total set continue decode false and quit(parity error). Otherwise do the procedure starting at step 8.3.16 to check the element widths in the character. If

they don't pass the tests, set continue decode false and quit (character elements out of limits).
v table (0 to 107)
(0) 6,6,6,4,4,4,4,4,4,4,
(10) 4,4,6,6,6,6,6,6,6,6,
(20) 6,6,6,8,6,6,6,6,6,6,
(30) 6,6,6,4,4,4,4,4,4,4,
(40) 4,4,6,6,6,6,6,6,8,6,
(50) 6,6,6,8,6,6,6,6,6,6,
(60) 8,4,6,4,4,4,4,4,4,4,
(70) 4,4,4,4,4,4,4,8,4,6,
(80) 6,6,6,6,6,6,6,6,6,8,
(90) 8,8,6,6,6,6,6,6,6,8,
(100) 8,8,8,4,4,6,6,6,

8.3.8 Compute the ratio of the sum of the elements in the current character to last char width. If this ratio is greater than max char ratio or less than min char ratio set continue decode false and quit.

8.3.9 If the length of label string is the maximum allowable label length set continue decode false and quit (label string overflow). Otherwise if current character is start (103,104,105) or stop (106) then go to 8.3.10. Otherwise set last char width to the width of the current character, add frame width to i, and append the current character to label string. Go to step 8.3.5.

8.3.10 Set continue decode false. If min margin ratio > element(i+8) / 2*(the sum of element(i+6) plus element(i+7)) then quit. Otherwise if forward is true then quit if either element(i+7)/element(i+1) > stop element ratio, or element(i+1)/element(i+7) > stop element ratio. Otherwise if forward is false reverse label string, and do the check sum calculation. If ok use the label string to construct the actual output string using the character table in the referenced spec. The values in label string give the entry for the "value" column in the table. Follow the rules to keep track of the current code when translating the data. The four function characters should be communicated to the control software for the decoder. If no errors are found set found label true. Quit.

8.3.11 This section is referenced from the main flow of the algorithm and you should return to the step which called this one when done here. The elements of the current character will be examined looking for a good character pattern. If forward is true go to step 8.3.12, otherwise go 8.3.13.

8.3.12 Find the total character width by adding elements i+1 through i+6. Calculate four two term sums T1 through T4 as follows.

T1 = element(i+1) + element(i+2)
T2 = element(i+2) + element(i+3)
T3 = element(i+3) + element(i+4)
T4 = element(i+4) + element(i+5)
Set bar total = (element(i+1) + element(i+3) + element(i+5))/total character width.
Go to 8.3.14.

8.3.13 Find the total character width by adding elements i+2 through i+7.
Calculate four two term sums T1 through T4 as follows.
T1 = element(i+6) + element(i+7)
T2 = element(i+5) + element(i+6)
T3 = element(i+4) + element(i+5)
T4 = element(i+3) + element(i+4)
Set bar total = (element(i+3) + element(i+5) + element(i+7))/total character width.

8.3.14 Compute five threshold values thresh 1, through thresh 5 by multiplying the total character width times threshold ratio 1 through threshold ratio 5.

8.3.15 Compute the four digits D1 through D4 of the pattern by doing the following for each sum T1 through T4. For j 1 through 4 do the following:
Dj = 2 if Tj < thresh 1.
Dj = 3 if thresh1 <= Tj < thresh 2.
Dj = 4 if thresh2 <= Tj < thresh 3.
Dj = 5 if thresh3 <= Tj < thresh 4.
Dj = 6 if thresh4 <= Tj < thresh 5.

Dj = 7 if thresh5 <= Tj.

Then pattern is equal to D4 + 16*D3 + 256*D2 + 4096*D1 (do this by shifting and adding as they are computed). the main algorithm.

8.3.16 This section is referenced from the main flow of the algorithm and you should return to the step which called this one when done here. This section checks the sizes of elements within a character for correct width ratios. Find the widest and narrowest bar and space.

8.3.17 If the ratio of widest bar to narrowest bar is greater than max element ratio then return; the test failed.

8.3.18 If the ratio of widest space to narrowest space is greater than max element ratio then return; the test failed.

8.3.19 Ok, return.

Section 9: UPC/EAN

9.1 General Code Description

A UPC/EAN label consists of one or more segments each representing a fixed number (4, 6 or 7) of numeric digits. This information is encoded in segments with a length of 4 or 6 characters. Each segment in the label can be scanned and decoded independently, without regard to order of segment capture or direction of scan.

9.1.1 Basic Structure

The widths of the alternating bars and spaces of the UPC/EAN label are defined in terms of modules. A module is a normalization factor used to compute element sizes when decoding the label. A module is generally the smallest of a group of elements making up specific sections of the label. All measurements and ratios used in the decoding algorithm are based on module widths.

9.1.2 Character Structure

Each character of a segment consists of a group of 4 elements, 2 bars and 2 spaces. Characters have a constant width of 7 modules with each element either 1, 2, 3 or 4 modules wide. Differing patterns of these 4 elements can be uniquely decoded to produce 20 characters, 0-9 both with even and odd parity. Parity is determined by the number of modules that make up the bars of the character pattern. The parity, and the direction in which the pattern was scanned (whether the character pattern starts with a space or bar), are used in conjunction with the segment to determine if the label had been scanned in a forward or backward direction.

9.1.3 Segment Structure

A segment comprises either 4 or 6 explicit characters. A seventh character is implicitly encoded into certain types of segments using the parity pattern. Segments are generally grouped into left and right halves. A left half segment and a right half segment are usually joined together by a center band. A center band is a pattern of 5 elements (3 spaces, 2 bars) that are each one module in width. The joined segments are surrounded on both sides with a margin (relatively large white space) and a guard bar pattern (3 one module elements, 2 bars, separated by one space). When scanned from left to right (forward direction), characters in the left half segment start with a space. The segment ends at the center band, and the right segment characters (scanned from the center band out) start with a mark (bar). Thus, the type of element that the character patterns in the segment start with determine the relative direction that the label was scanned (margin to center band, or center band to margin). The parity pattern generated by the segment characters is used to identify a left or right half segment. These two pieces of information identify the type

of segment and order of the encoded data. Certain types of left half segments are not joined by corresponding right halves. These segments have a left margin and guard bar pattern and a center band pattern on the right. The center band contains an extra 1 module bar on the right to separate the last space from the margin.

9.1.4 Label Structure

Specific label types are formed with 1 or more unique segments. UPC type A labels have a six character left and right half for a total of 12 characters. UPC type E (zero suppressed) labels have 1 six character left half with a modulo 10 check digit encoded as a seventh character in the parity pattern. EAN-13 labels have 2 six character halves, with a 13th digit encoded in the left half segment parity pattern. EAN-8 labels have one left and one right 4 character half, for 8 digits of encoded data. UPC type D labels (D-1 thru D-5) have various combinations of both 4 and 6 character segments creating labels with 14 to 32 digits of data. Segment halves or pairs are separated within the label by intra-block and inter-block gap elements. These are essentially margin elements with a minimum nominal width of 7 modules.

9.1.5 Supplemental Addon Encodation

UPC A and E, EAH13, and EAN8 labels can be suffixed past the right margin by supplementally encoded data (also called periodical data, because it is used primarily by the magazine and book industry) of length 2 or 5 characters. The data immediately follows the right margin (7 modules nominal) and is deliniated by a guard bar pattern of 3 elements (1 module bar, 1 module space, 2 module bar). Character patterns are separated by 2 elements (1 module space, 1 module bar), and a margin element immediately follows the last character. The supplemental encoding scheme was purposely designed so as to not interfere with the decoding of the standard UPC/EAN label, but also with the ability to utilize some of the already existing decoding schemes (such as character pattern recognition) in a UPC/EAN decoder. Because of its nature, the supplemental encodation is more susceptible to error than standard UPC/EAN label types.

CHARACTER	EVEN PARITY		ODD PARITY	
	$B_1S_1B_2S_2$	T_1T_2	$B_1S_1B_2S_2$	T_1T_2
0	3211	53	1123	23
1	2221	44	1222	34
2	2122	33	2212	43
3	1411	55	1141	25
4	1132	24	2311	54
5	1231	35	1321	45
6	1114	22	4111	52
7	1312	44	2131	34
8	1213	33	3121	43
9	3112	42	2113	32

Figure 9.1:
Bar (B_i) Space (S_i) and 2-term sum (T_i) information for each UPC/EAN character. When scan direction is such that character starts with a space, then character elements should be reversed. Note that segments containing both odd and even parity characters will contain reversed characters.

9.2 Decoding Algorithm Overview

9.2.1 Decode Status Variables

Variables used in the decoding process are defined below. Values of the status variables can be maintained globally to allow re-entrant use of the decoder.

Variable name	Description
i	pointer to the current element in the data buffer. This always points to a space when the decoder is called.
ia	pointer to the current element being analyzed for supplemental addon data.
current margin	pointer to the last space that was determined to be a margin.
last decode point	pointer to the last point where data in the data buffer was either decoded or rejected as bad. This pointer limits decode attempts in the backward direction to data where no attempt to decode has taken place.
first buffer element	pointer to oldest valid data in the data buffer.
last char width	sum of the element widths in the last character decoded in the current segment.
label string	decoded characters of a fully formed label stored as an ASCII string.
segment string	decoded characters of a segment stored as an ASCII string. These are accumulated in the segment store.
addon string	decoded characters of the addon segment stored as an ASCII string.
fwd data available	boolean; true if the buffer has more elements to decode in the forward direction.
continue decode	boolean; true when decoding and the buffer at the current element looks like good data.
fwd decode	boolean; true when attempting decode of the buffer in forward direction (false if backwards).
bkwd data available	boolean; true if the buffer has more elements to decode in the backward direction.
addon data available	boolean; true if the buffer has more elements to decode for the addon.

continue decode addon	boolean;	true when decoding in the addon and the buffer up to the current element looks like good data.
fwd decode addon	boolean;	true when attempting decode of the buffer in forward direction (false if backwards).
found segment	boolean;	true when a segment has been put into the segment buffer.
reverse	boolean;	true if a segment was decoded from center-band out to the margin.
found addon	boolean;	true when an addon segment has been put into the addon buffer.
found label	boolean;	true when a label has been put into label string.
parity bits		array of six bits representing the parity pattern of the segment decoded. A 1 bit indicates EVEN parity.
addon parity bits		array of 2 or 5 bits representing the parity pattern of the addon segment decoded. A 1 bit indicates even parity.
segment type		variable indicating the type of segment in the segment buffer (UPC-A right, EAN-13 left, etc.).
segment store		array of buffers storing decoded segments as ascii strings.
data buffer		array of numbers representing widths of the elements scanned. They must alternate between bars and spaces.

Note: It is possible in an actual implementation of this algorithm to use one variable to represent several of the above. The variables were separated to allow for more clarity in the algorithm descriptions.

9.2.2 Decode Constants

Constant values are identified in this section. The constants are referenced by name in the description of the algorithm

Constant	Value	Description
frame width	4	Number of elements in a character
threshold 1,2,& 3	25/70 35/70 45/70	Decision points used to base weightings of the 2-term sums expressed as ratio of sum to character width.
ambiguity scale	2/3	Utilized to scale a character element width for ambiguity resolution.
max char ratio	1.25	Max ratio of the sums of the elements in the current character to the previous character.
min char ratio	.80	1/max char ratio is the min char ratio
margin scaler	1.75	Factor used to scale margin guard bar sum (2 bar plus 2 times the space), to make it comparable to a character width.
max margin char ratio	2	Max ratio of the sums of the elements in the first character of the segment to the margin (margin is multiplied by the margin scaler constant). 1/max margin char ratio is the min margin char ratio.
min margin ratio	1.375	Minimum ratio of a white space to the sum of the next 2 bars plus 2 times the next space (next three elements, either forward or backward) to qualify it as a margin element.
max margin ratio	3	Max ratio of a white space to the sum of the next 2 bars plus 2 times the next space (next 3 elements, either forward or backward) to qualify it a legal gap between label segment and add-on segment.
max like element ratio	1.5	Max ratio of two 1 module wide elements when both are bars or both are spaces.
max addon guard bar ratio, min addon guard bar ratio	2.5 1.5	Max and min ratio of the second bar (third element) to the first bar (first element) of an addon segment.

9.2.3 Derivation of the constants

threshold 1,2, & 3

Detail on the choice of these numbers is descibed in section 9.2.4.1. Basically, it is the midpoint between nominal ratios of the 2-term sums of the character to sum of all elements in the character (character width). Since nominal ratios are 2/7, 3/7, 4/7, and 5/7, the thresholds are set as 2.5/7, 3.5/7, and 4.5/7.

ambiguity scale

This is used as a constant to scale l_2 as described in section 9.2.4.1 when resolving EVEN 1,7 ambiguity. Since the possible ratios of l_1 to l_2 are 1/3 and 1, scaling l_2 by 2/3 (.6667) results in ratios of 1/2 and 3/2. Choosing ratio of 1 as decision point is near midpoint between these two extremes, and allows simple magnitude comparison of the elements (ie. which is greater).

max char ratio

This value is chosen based on the possible variation in spot velocity and label printing tolerances. 1.25 is used until more information becomes available on the scanning device used. For example, rotating polygon scanners generally have less spot speed variation than resonant scanners. Resonant scanners would, therefore need a larger range of character ratios (larger max char ratio constant) than polygon scanners.

margin scaler

This number is used to calculate a "character width" for the margin guard band. The measured width (bars + 2 * space) is nominally 4 modules. Multiplication by 7/4 (1.75) gives a pseudo width of 7 modules (which is a nominal character width). This number can then be compared directly with the subsequent character in the label.

max margin char ratio

Since the margin "character width" is calculated rather than measured directly, its accuracy is lessened. This value (2.0) was chosen so as to be more lenient than max char ratio, but still allow for relative comparisons of margin guard bars to characters in a segment.

min margin ratio

A nominal margin white space is 7 modules wide. The largest element within a label is 4 modules. Choosing the mid-point as the decision point gives a value of 5.5 modules. The margin ratio is calculated using the bars of the guard band plus 2 times the space. For nominal elements, this would be 4 modules. The threshold point is therefore set at 5.5/4 = 1.375.

max margin ratio

This value is used to limit the size of the gap between the supplemental addon segment and the standard UPC/EAN segment that it attaches to. Addon data is fairly easily made from scan data that is not part of the label. The upper limit of the gap size helps prevent use of data that is not part of the label from making an invalid addon. The sum of the widths of the guard band of an addon segment is 5 modules (the space is added twice). Using a max margin ratio of 3.0 allows for gaps of up to 15 modules. Since nominal gap size is 7 modules, this gives ample room for deviation in a valid label.

max like element ratio

This value is used verify that two 1 module like elements are the same. Next biggest element is 2 modules, therefore the midpoint (1.5) is chosen to be the best decision point for this test.

max addon guard bar ratio, min addon guard bar ratio

The basis for these numbers is similar to above, with the exception that the guard bar ratio is nominally 2 (the second bar is 2 modules wide nominally). Values of 1.5 and 2.5 are midpoints in the decision space between ratio values of 1, 2 and 3.

9.2.4 General Decoding Method

9.2.4.1 Character Decoding

The UPC/EAN decoder can use "like-edge" measurements for most of the character recognition functions. That is to say, measurements are made from the leading (trailing) edge of a bar to the leading (trailing) edge of the next bar. The decoder sums the widths of the individual elements in a bar-space pair to determine the appropriate like-edge measurement, called a 2-term sum. For a UPC character, two 2-term sums are calculated, T_1 and T_2. These are defined as the sums of the first bar, first space and first space, second bar respectively. The definitions and subsequent ones assume that the first element of the character being decoded is a bar. If the first element is a space, the order of the 4 elements of the character should be reversed. The sums T_1 and T_2 generate values ranging from 2 to 5 modules. This provides the capability to distinguish between 16 possible combinations of the sums. (See figure 9.1) 12 of the 16 combinations result in unique UPC/EAN character determination. The remaining 4 indicate ambiguous character pairs (even and odd parity 1,7 and 2,8). The two term sums do not give sufficient information to uniquely identify these characters, so more measurements are needed. The nature of these measurements are discussed later in this section. Note that the last space of the character is not used for generation of 2-term sums. This is because the error tolerance on the ending space of the character is significantly larger than the other elements of the character. To determine the size in modules of each 2-term sum, it is normalized by first dividing it by the total character width (T). Since the total width is defined to be 7 modules, module size of each of the sums is in terms of 1/7th's of the total width. Given these calculations, the following decision rules can be established with respect to the interpretation of T_i:

$$\frac{T_1}{T} \le \frac{X_1}{7} \quad \text{--> } T_1 \text{ is 2 modules}$$

$$\frac{X_1}{7} < \frac{T_1}{T} \le \frac{X_2}{7} \quad \text{--> } T_1 \text{ is 3 modules}$$

$$\frac{X_1}{7} < \frac{T_1}{T} \le \frac{X_3}{7} \quad \text{--> } T_1 \text{ is 4 modules}$$

$$\frac{X_3}{7} < \frac{T_1}{T} \quad \text{--> } T_1 \text{ is 5 modules}$$

where X_i is the appropriate decision threshold in terms of modules.

Choosing values for X_i that are at the midpoint in the decision space yields the following:

$$X_1 = 2.5 \;\text{-->}\; \frac{X_1}{7} = 0.3571 = \text{Threshold 1}$$

$$X_2 = 3.5 \;\text{-->}\; \frac{X_2}{7} = 0.5 = \text{Threshold 2}$$

$$X_3 = 4.5 \;\text{-->}\; \frac{X_3}{7} = 0.6429 = \text{Threshold 3}$$

As mentioned earlier, more information is required to determine the correct character of an ambiguous pair for 4 combinations of T_1 and T_2. T_1 and T_2 provide enough information to determine parity, and the specific ambiguous character pair (1,7 or 2,8). Resolving the ambiguity can be accomplished by utilizing the first space and second bar of the character (the second and third elements of the character, respectively) or the first bar and first space (first and second elements). Calculating the ratio of the third element to the second element(I_3/I_2) and the first element to the second element(I_1/I_2) yields the following:

Character	I_3/I_2	I_1/I_2
EVEN 1	1	1
EVEN 7	1/3	1/3
ODD 1	1	1/2
ODD 7	3	2
EVEN 2	2	2
EVEN 8	1/2	1/2
ODD 2	1/2	3
ODD 8	2	1

Note that, with the exception of EVEN 1 7, resolution of every ambiguous pair required only a determination of which element is larger. Even 1 7 requires a scale factor before comparison. A midpoint selection yields a factor of 2/3. Thus the decision rules for resolving character ambiguity are:

```
EVEN 1 7 :  I2 • X4 > I1   -->  EVEN 7   else  EVEN 1
ODD  1 7 :  I2 > I1        -->  ODD  1   else  ODD  7
EVEN 2 8 :  I3 > I2        -->  EVEN 2   else  EVEN 8
ODD  2 8 :  I3 > I2        -->  ODD  8   else  ODD  2
```

Where $X_4 = 2/3 = 0.6667$

9.2.4.2 Segment Decoding

Proper framing requires that a segment be decoded from the margin to the center band or, if the decoder has just previously recognized a segment ending on the center band, from the center band to the margin. In cases where the scanner has passed through part of a segment, the center band, and the entire adjoining segment, the decoding will be performed in a backward direction, from the margin just encountered back to the center band. As the character recognition method described above will always return a "valid" character, some other means of rejecting data that is not truly valid must be utilized. The primary method utilized is measurement of relative widths between characters of the segment. Secondary checks of valid parity patterns are done if the character width check passes.

9.2.4.3 Label Decoding

As previously discussed, the segments of a UPC/EAN label can be scanned in random order. The only exception is supplemental addon data, which must be decoded in conjunction with a valid UPC-A or EAN-8 right half or UPC-E segment. Labels are reconstructed from the decoded segments by utilizing the parity pattern of the encoded characters in each segment. Figures 9.2 and 9.3 identify the valid parity patterns for label segments, and the allowed combinations of segments making up the various labels. Many labels contain some of the same segments. The decoder should attempt to assemble labels in order of diminishing complexity, to minimize segment substitution errors. For example, the decoder should start by attempting a UPC D-5 label first since it contains the most segments. If unsuccessful, a D-4 should be tried, and so on.

NOTATION	USED IN	PARITY PATTERN*
N(1)	EAN-13	OOEOEE
N(2)	"	OOEEOE
N(3)	"	OOEEEO
N(4)	"	OEOOEE
N(5)	"	OEEOOE
N(6)	"	OEEEOO
N(7)	"	OEOEOE
N(8)	"	OEOEEO
N(9)	"	OEEOEO
E(0)	UPC-E	EEEOOO
E(1)	"	EEOEOO
E(2)	"	EEOOEO
E(3)	"	EEOOOE
E(4)	"	EOEEOO
E(5)	"	EOOEEO
E(6)	"	EOOOEE
E(7)	"	EOEOEO
E(8)	"	EOEOOE
E(9)	"	EOOEOE
A(L)	UPC-A	OOOOOO
A(R)	UPC-A, EAN-13, VERS-D	EEEEEE
D	VERS-D	OOOEEE
n(1)	"	EEOO
n(2)	"	OEEO
n(3)	"	EOOE
n(4)	"	EOEO
n(5)	"	OEOE
n(6)	"	OOEE
8(L)	EAN-8, VERS-D	OOOO
8(R)	" "	EEEE

Figure 9.2 Segment Parity Patterns

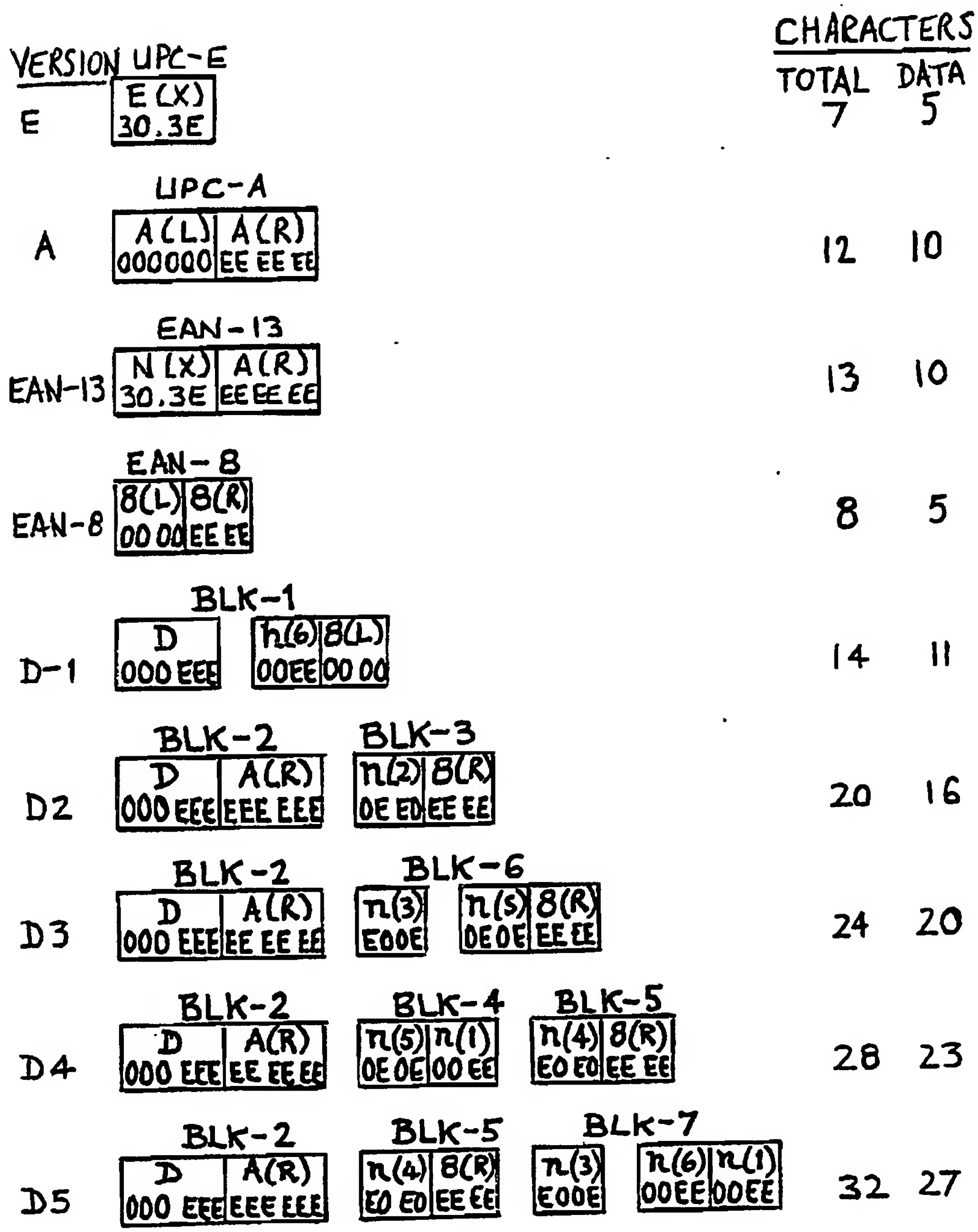


Figure 9.3 Valid Label structures

9.3 UPC/EAN Decode Algorithm

The specific decoding algorithm is given below. Before entering the UPC algorithm the first time the segment store array should be cleared and last decode point set to first buffer element. The pointer i should be set to a large white space. When decoding in the forward direction, failure of any test results in an exit from the algorithm with continue decode variable set to false. When decoding in the reverse direction, failure of any test should set continue decode false, set the variable last decode point and the current element pointer (i) to the current margin (to prevent another search backward), and re-start the algorithm.

9.3.1 If scanning continuously and no segments have been found for a while, or if the trigger was just pulled, clear the segment buffer. This should be managed in a way that prevents combining segments from different items and depends on characteristics of the scan head used.

9.3.2 Look for a margin backward:
If the difference between i and the last decode point is equal to or greater than a minimum segment size (4 char segment = 26 elements) the do step 9.3.10 to check for a margin in the reverse direction. If a margin has been found then:
- set last char width to margin scaler * (element(i+1) + 2*element(i+2) + element(i-3).
- set continue decode to true.
- set current margin to i.
- set i to i +frame width (to point to first element of next character to decode).
- set fwd decode false.
- reset the parity bits and segment string to empty.

9.3.3. If continue decode is false (didn't find a margin backward), then look for a margin forward: if there are less than frame width elements available to be examined forward, wait (not enough elements to find margin pattern). If enough, then do step 9.3.10 to check for margin in forward direction. If a margin is found then:
- set last char width to margin scaler * (element(i+1) + 2*element(i+2) + element(i+3).
- set continue decode to true.
- set current margin to i.
- set i to i +frame width (to point to first element of next character to decode).
- set fwd decode true.
- reset the parity bits and segment string to empty.

9.3.4 If continue decode is false then quit (didn't find margin either way).

9.3.5 If continue decode is true and fwd decode is false, do step 9.3.11 to look for a valid segment in the backward direction. If continue decode is true and fwd decode is true, then do step 9.3.16 to look for valid segment in the forward direction.

9.3.6 If segment found, check for addon by doing step 9.3.26.

9.3.7 This step deleted.

9.3.8 Add the previously captured segment string to the segment store array. Set found segment to false, and set segment string to empty.

9.3.9.1 If continue decode is true and fwd decode is true do step 9.3.16 (Try to read the next segment of a label.)

9.3.9.2 Try to make a label by doing step 9.3.38. If make label was successful, then set found label true, set label type.

9.3.9.3 Return

9.3.10 Margin check algorithm:
If fwd decode true, check that element(i)/(element(i+1) + 2 :* element(i+2) + element(i+3)) > min margin ratio. If fwd decode false, check that element(i)/(element(i-1) + 2*element(i-2) + element(i-3)) > min margin ratio. If not ok, then test failed, return. If ok, then if fwd decode true calculate ratio = element(i+1)/element(i+3). If fwd decode false, ratio = element(i-1)/element(i-3). If ratio > max like element ratio or (1/ratio) > max like element ratio then test failed, return. Otherwise, test passed, return.

9.3.11 Get backward segment algorithm:
Check if data is available in the backward direction (i >= last decode point +1). If not available, set continue decode false and go to step 9.3.14.

9.3.12 Use steps 9.3.21 thru 9.3.24 to get a possible segment character. If successful:
- add character to the segment string.
- shift the parity bits map left 1 bit.
- add 1 to the map if the character is even parity.

- set last char width to the current char width
- subtract frame width from i

Otherwise, go to step 9.3.14.

9.3.13 If length of segment string greater than 6 then set continue decode false and go to step 9.3.15 (too many characters in the segment string). Otherwise, go back to step 9.3.11.

9.3.14 If length of segment string is not 4 or 6 then goto step 9.3.15. Otherwise, if element(i) is not a bar then go to step 9.3.15 (framing problem). Otherwise, if char width was not too small then go to step 9.3.15 (isn't a center-band). Otherwise, if the decoded character is not an ambiguous character (1,2,7, or 8) then go to step 9.3.15 (not a center-band). Otherwise, set last char width to current width, decrement i by 1 and use steps 9.3.21 thru 9.3.24 to get another character. If not successful or the decoded character is not an ambiguous character then go to step 9.3.15. Otherwise, use step 9.3.25 to decode segment parity map with reverse set false. If segment type is ok, then set segment found true and check if segment type is UPC-A right half or EAN-8 right half or UPC D n1, reverse segment string digits if segment is one of those types. (scanned it backward)

9.3.15 Set last decode point to the current margin. Set continue decode to false and set i back to current margin. Return to main section of the algorithm.

9.3.16 Get forward segment algorithm:

Check if data is available in the forward direction (i <= last buffer element + frame width). If not available, wait.

9.3.17 Use steps 9.3.21 thru 9.3.24 to get a possible segment character. If successful:

- add character to the segment string.
- shift the parity bits map left 1 bit.
- add 1 to the map if the character is even parity.
- set last char width to the current char width
- add frame width to i

Otherwise, go to step 9.3.19.

9.3.18 If length of segment string is less than 6 then go to step 9.3.16. Otherwise set continue decode to false and return.

9.3.19 Set continue decode to false. If length of segment string is not 4 or 6 then return. Otherwise, if element(i) is a bar then go to step 9.3.20. Otherwise, if char width was not too small then return (isn't a center-band). Otherwise, if the decoded character is not an ambiguous character (1,2,7, or 8) then return (not a center-band). Otherwise, set last char width to current width, increment i by 1 and use steps 9.3.21 thru 9.3.24 to get another character. If not successful or the decoded character is not an ambiguous character then return. Otherwise, use step 9.3.25 to decode segment parity map (with reverse flag false). If segment type not ok return, otherwise set found segment true and check if segment type is UPC-A right half or EAN-8 right half or UPC D n1, reverse segment string digits if segment is one of those types. (scanned it backward). Then set continue decode true (to try decoding from the center band out), set i to i + frame width, and return.

9.3.20 Segment ended on margin:

If character is not too big then return (not a margin char). Increment i by 3, set fwd decode false, and do step 9.3.10 to check margin. Set fwd decode true. If margin not OK return, otherwise use step 9.3.26 to decode segment parity map (with reverse flag true). If segment type not ok return, otherwise set found segment true and check if segment is not type UPC-A right half or EAN-8 right half or UPC D n1, if it isn't then reverse segment string (scanned a left half backward). Do step 9.3.8. Set last decode point to i, set current margin to i. Look for an addon by doing step 9.3.26. Go to 9.3.3 to try finding additional label segments.

9.3.21 Get character:

Sum elements from i to i+3 to get char width. If element(i) is a bar, then set 2-term sum 1 to element(i)+element(i+1), 2-term sum 2 to element(i+1)+element(i+2). Otherwise, set 2-term sum 1 to element(i+2)+element(i+3), 2-term sum 2 to element(i+1)+element (i+2). Set ratio1 to 2-term sum 1 / char width, ratio2 to 2-term sum 2 / char width. If ratio1 < threshold 1 then set weight to 0, otherwise if ratio1 < threshold 2 then set weight to 4, otherwise if ratio1 < threshold 3 then set weight to 8, otherwise set weight to 12. If ratio2 < threshold 1 then do nothing, otherwise if ratio2 < threshold 2 then set weight to weight+1, otherwise if ratio2 < threshold 3 then set weight to weight+2, otherwise set weight to weight+3.

9.3.22 Use the value weight to index into the following table of characters:

0: character = 6, even=true;
1: character = 0, even=false;

2: character = 4, even=true;
3: character =3, even=false;
4: character = 9, even=false;
5: character = 2, even=true;
6: character = 1, even=false;
7: character = 5, even=true;
8: character = 9, even=true;
9: character = 2, even=false;
10: character = 1, even=true;
11: character =5, even=false;
12: character =6, even=false;
13: character = 0, even=true;
14: character = 4, even=false;
15: character = 3, even=true;

For example, if weight=4, then the decoded character = 9 and even parity is set false.

9.3.23 Check ambiguities:

If character = 2 and element(i) is a space and even is true, and element(i+2) > element(i+1) then set character to 8.

If character = 2 and element(i) is a bar and even is true, and element(i+2) < element(i+1) then set character to 8.

If character = 2 and element(i) is a space and even is false, and element(i+2) < element(i+1) then set character to 8.

If character = 2 and element(i) is a bar and even is false, and element(i+2) > element(i+1) then set character to 8.

If character = 1 and element(i) is a space and even is false, and element(i+3) > element(i+2) then set character to 7.

If character = 1 and element(i) is a bar and even is false, and element(i) > (i+1) then set character to 7.

If character = 1 and element(i) is a space and even is true, and element(i+2)*ambiguity scale > element-(i+3) then set character to 7.

If character = 1 and element(i) is a bar and even is true, and element(i+1)*ambiguity scale > element(i) then set character to 7.

9.3.24 Check widths:

If char width / last char width > max char ratio then return with char too big indication. If char width / last char width < (1/min char ratio) then return with char too small indication. Otherwise, return with successful indication.

9.3.25 Segment parity decode:

If reverse is true, then reverse order of parity bits. If the length of the segment string is 6, then set segment type and encoded digit from the following look up table:

```
$00 (000000), segment type = UPC A L,   encoded digit = 0
$07 (000EEE), segment type = UPC D,     encoded digit = 0
$0B (00E0EE), segment type = EAN13 L,   encoded digit = 1
$0D (00EE0E), segment type = EAN13 L,   encoded digit = 2
$0E (00EEE0), segment type = EAN13 L,   encoded digit = 3
$13 (0E00EE), segment type = EAN13 L,   encoded digit = 4
$15 (0E0E0E), segment type = EAN13 L,   encoded digit = 7
$16 (0E0EE0), segment type = EAN13 L,   encoded digit = 8
$19 (0EE00E), segment type = EAN13 L,   encoded digit = 5
$1A (0EE0E0), segment type = EAN13 L,   encoded digit = 9
$1C (0EEE00), segment type = EAN13 L,   encoded digit = 6
$23 (E000EE), segment type = UPC E,     encoded digit = 6
$25 (E00E0E), segment type = UPC E,     encoded digit = 9
$26 (E00EE0), segment type = UPC E,     encoded digit = 5
$29 (E0E00E), segment type = UPC E,     encoded digit = 8
$2A (E0E0E0), segment type = UPC E,     encoded digit = 7
$2C (E0EE00), segment type = UPC E,     encoded digit = 4
$31 (EE000E), segment type = UPC E,     encoded digit = 3
$32 (EE00E0), segment type = UPC E,     encoded digit = 2
$34 (EE0E00), segment type = UPC E,     encoded digit = 1
$38 (EEE000), segment type = UPC E,     encoded digit = 0
$3F (EEEEEE), segment type = UPC A R,   encoded digit = 0
```

For example, a parity bits map of 2A hex would set segment type to UPC E and encoded digit to 7. If segment type = UPC E then add encoded digit to end of segment string. If segment type = EAN 13 L then add encoded digit to start of segment string.

If length of segment string = 4 then set segment type and encoded digit from the following table:

```
$00 (0000), segment type = EAN8 L,     encoded digit = 0
$03 (00EE), segment type = UPC D n6,   encoded digit = 6
$05 (0E0E), segment type = UPC D n5,   encoded digit = 5
$06 (0EE0), segment type = UPC D n2,   encoded digit = 2
$09 (E00E), segment type = UPC D n3,   encoded digit = 3
$0A (E0E0), segment type = UPC D n4,   encoded digit = 4
$0C (EE00), segment type = UPC D n1,   encoded digit = 1
$0F (EEEE), segment type = EAN8 R,     encoded digit = 0
```

If the parity bits map is not contained in the table, set error indication and return.

9.3.26 Check for supplemental addon segment:

Set continue decode addon to false. If found addon has not been set true, then check current segment found. If current segment type is UPC E and both forward decode and reverse are false (segment was decoded backwards from margin to center band), then set is to the current margin - 34 (width of an E-segment), fwd decode addon to false, and continue decode addon to true.

If current segment type is UPC E and fwd decode is true and reverse is false (segment was decoded forward from margin to center-band), then set is to the current margin + 34 (width of an E-segment), fwd decode addon to true, and continue decode addon to true.

If current segment is UPC A Right or EAN 8 right, and both fwd decode and reverse are true (forward direction decode from center-band to margin), then set is to the i, set fwd decode addon true, and set continue decode addon true.

If current segment is UPC A Right or EAN 8 Right, and fwd decode is true but reverse is false (forward direction decode from margin to center-band), then set is to current margin, fwd decode addon to false, and continue decode addon true.

If current segment is UPC A Right or EAN 8 Right, and both fwd decode and reverse are false (backward decode from margin to center-band), then set is to current margin , fwd decode addon to true and continue decode addon to true.

9.3.27 If continue decode addon is true (found a segment with a posisible addon) then clear addon parity bits, addon string. If continue decode addon is true and fwd decode addon is true then set addon data available to true. If less than a frame width worth of elements are in the data buffer, wait (ai greater than last buffer location - frame width). If continue decode addon is true and fwd decode addon is false then set addon data available to true if ai is greater than or equal to 13 + first buffer element(at least enough to make 2 char addon), otherwise set it false. If addon data available is set false, then set continue decode addon to false and return (not enough data backward to make addon segment). Otherwise, if fwd decode addon is false go to 9.3.31.

9.3.28 Decode addon in forward direction:
If addon string empty, then wait until at least 2*frame width counts are available in the buffer (ai< = last element - 2 + frame width) else (if addon string not empty) then wait until at least frame width counts are available in the buffer (ai< = last element - (frame width + 1).

9.3.29 Do all of this section if addon string empty.
Check that element(ai)/(element(ai + 1) + element(ai + 2) + element(ai + 3)) is greater than min margin ratio and less than max margin ratio. If ok, then check that element(ai + 3)/element(ai + 2) is greater than min addon guard bar ratio and less than max addon guard bar ratio. If ok, then check that element(ai + 2)-/element(ai + 1) is less than max like element ratio and greater than (1/max like element ratio). If ok, then set addon last char width to (element(ai + 1) + 2*element(ai + 2) + element(ai + 3)) * 9/5, set ai to ai + frame width. If any test not ok, set continue decode addon false.

9.3.30 If addon data available is true and continue decode addon is true, then do steps 9.3.21 to 9.3.23 to get character (don't check widths), else return. If addon string is empty then add element(ai-3)-+element(ai-2) to char width, else add element(ai-2) + element(ai-1) to char width. If char width / addon last char width > max char ratio or < (1/max char ratio) then set continue decode addon to false, else append decoded character to addon string, set ai to ai + frame width + 2, set last char width to char width, append parity bit (even = 1) to addon parity bits, set continue decode addon false if addon string has 5 elements (maximum). Go to step 9.3.35

9.3.31 Decode addon in backward direction:
If addon string empty, then set addon data available to true if enough data to make guard bar plus first char (ai> = first buffer element + 2 * frame width) else (if addon string not empty) then set addon data available to true if at least frame width counts in buffer (ai> = first buffer element + frame width).

9.3.32 Do all of this section if addon data available is true and addon string empty.
Check that element(ai)/(element(ai-1) + element(ai-2) + element(ai-3)) is greater than min margin ratio and less than max margin ratio. If ok, then check that element(ai-3)/element(ai-2) is greater than min addon guard bar ratio and less than max addon guard bar ratio. If ok, then check that element(ai-2)/element(ai-1) is less than max like element ratio and greater than (1/max like element ratio). If ok, then set addon last char width to (element(ai-1) + 2*element(ai-2) + element(ai-3)) * 9/5, set ai to ai minus (2 + frame width - 1). If any test not ok, set continue decode addon false.

9.3.33 If addon data available is true and continue decode addon is true, then do steps 9.3.21 to 9.3.23 to get character (don't check widths), else go to step 9.3.35. If addon string is empty then add element(ai + 5) + element(ai + 6) to char width, else add element(ai + 5) + element(ai + 4) to char width. If char width / addon last char width > max char ratio or < (1/max char ratio) then set continue decode addon to false, else append decoded character to addon string, set ai to ai-frame width-2, set last char width to char width, append parity bit (even = 1) to addon parity bits, set continue decode addon false if addon string has 5 elements (maximum).

9.3.34 If addon data available is false then set continue decode addon to false and return to the main algorithm (not enough backward data available).

9.3.35 If continue decode addon is true then if fwd decode addon is true then go to step 9.3.28. If continue decode addon is true and fwd decode addon is false, then go to step 9.3.31. Otherwise, if length of addon string not equal to 2 or 5 then return (not a valid addon).

9.3.36 If length of addon string is 2, then calculate parity as the mod 4 of the value of the 2-digit addon (eg. if addon string is '13', then parity is 13 mod 4 = 1). If calculated parity doesn't match parity bits then quit. Otherwise set found addon to true, and addon type to 2-char, and return.

9.3.37 If length of addon string is 5, then calculate parity as (3 * sum of digits 1, 3 and 5 of addon string + 9 * sum of digits 2 and 4) mod 10. Index the parity digit into the following table (starting reference of 0):
($18, $14, $12, $11, $0C, $06, $03, $0A, $09, $05)
If the indexed parity pattern equals addon parity bits then set found addon to true, and addon type to 5-char, then return.

9.3.38 Attempt to assemble label:

(There are many possible methods of deciding validity of a label. The one presented here checks for capture of segments needed to make labels as follows: If a UPC-D segment was seen, try UPC-D5 through D1. If no D segment was seen, try UPC-A, then EAN-13, then UPC-E, then EAN-8. Check digits are tested in each block as it is evaluated. The segment capture and label assembly methods used in this algorithm are straightforward.

More sophisticated methods may be used. One method would be to keep a count of how many times a particular segment has been seen, and use a rule controlling when it would be replaced by newly scanned data. For example, if a particular UPC-A-R segment was seen one time, then a different one was seen, replace the old one in the segment array with the new one. If the old one had been seen two or more times, keep the old one, and discard the new one. This makes it easy to require seeing certain error prone segments (UPC-E, EAN-8) twice by just requiring a total of two or more. Another method is to keep two complete segment array buffers, each with a set of totals for how many times each segment has been seen. Initially, the buffers and totals are all cleared. Up to two different versions of each segment type are collected, with totals for how many times they have been seen. If a third version of a particular segment type is seen, it is discarded. Then during the process of determining if a good label has been seen, the two counts for each segment type are examined. If one version of the data has been seen much more than another, it is accepted. If two versions of the data have been seen frequently, it is not accepted. For example, if a UPC-D segment 012345 was seen three times, and a UPC-D segment 018345 was seen two times, don't accept either. If the first one was seen four times, and the second one was seen one time, accept the first as being ok.)

9.3.38.1 If segment store array contains UPC-D go to 9.3.38.2, otherwise go to 9.3.38.10.

9.3.38.2 If segment store array contains UPC-A-R then go to 9.3.38.4, otherwise go to 9.3.38.3.

9.3.38.3 If segment store array contains UPC-D-n6, EAN-8-L, and the Block 1 checksum calculation is ok (step 9.3.39) set label type to UPC-D1, set found label true. Return in all cases.

9.3.38.4 If the Block 2 checksum calculation is not ok (step 9.3.39) return. Otherwise go to 9.3.38.5.

9.3.38.5 If segment store array contains UPC-D-n4 and EAN-8-R, do Block 5 checksum calculation (step 9.3.39). If ok go to 9.3.38.6, otherwise go to 9.3.38.8.

9.3.38.6 If segment store array contains UPC-D-n3 and UPC-D-n6 and UPC-D-n1 and the Block 7 checksum calculation is ok (step 9.3.39) set label type to UPC-D5, set found label true, return. Otherwise go to 9.3.38.7.

9.3.38.7 If segment store array contains UPC-D-n5 and UPC-D-n1 and the Block 4 checksum calculation is ok (step 9.3.39), set label type to UPC-D4, set found label true, return. Otherwise go to 9.3.38.8.

9.3.38.8 If segment store array contains UPC-D-n3 and UPC-D-n5 and Ean-8-R and the Block 6 checksum calculation is ok (step 9.3.39), set label type to UPC-D3, set found label true, return. Otherwise go to 9.3.38.9.

9.3.38.9 If segment store array contains UPC-D-n2 and Ean-8-R and the Block 3 checksum calculation is ok (step 9.3.39), set label type to UPC-D2, set found label true. Return in all cases.

9.3.38.10 If segment store array contains UPC-A-R, go to 9.3.38.11. Otherwise go to 9.3.38.13.

9.3.38.11 If segment store array contains UPC-A-L, and the UPC-A checksum calculation is ok (step 9.3.39), set found label true, set label type to UPC-A, return. Otherwise go to 9.3.38.12.

9.3.38.12 If segment store array contains UPC-13-L and the EAN-13 checksum calculation is ok (step 9.3.39), set label type to EAN-13, set found label true. Return in all cases.

9.3.38.13 If segment store array contains UPC-A-L return. Otherwise, go to 9.3.38.14.

9.3.38.14 If segment store array contains UPC-E, and the UPC-E checksum calculation is ok (step 9.3.39), set label type to UPC-E, set found label true, return. If no UPC-E was in the buffer, go to 9.3.38.15, otherwise return.

9.3.38.15 If segment store array contains UPC-8-L and EAN-8-R, and the EAN-8 checksum calculation is ok (step 9.3.39), set label type to EAN-8, set found label true. Return in all cases.

9.3.39 Checksum calculation:

Calculate the check sum of a block using the UPC specification rules. See figure 9.3 for a definition of the various blocks. If any checksums are not zero, then the test failed.

The checksum for each block is calculated by starting from the rightmost character of a block, adding the numeric value of the characters multiplied by a weighting factor. The weighting factor is alternately 3 and 1. If the mod 10 sum of the weighted characters is 0, the checksum is correct.

Example: For a Block 6 (found in a UPC D-3) with characters 123456789104, the sum is 3x4 + 1x0 + 3x1

+ 1x9 + 3x8 + 1x7 + 3x6 + 1x5 + 3x4 + 1x3 + 3x2 + 1x1 = 100. 100 mod 10 = 0, so it is ok.

A program listing for the above algorithms in assembly language is given below for implementation on a Thompson-Mostek MK68HC200 microprocessor.

```
;*************************************************************************
;*                                                                       *
;*                          DECODE68                                     *
;*                                                                       *
;*      68200 minimal operating system, and application code for        *
;*      code 3 of 9 and the various upc/ean codes written in a          *
;*      reduced instruction set.                                        *
;*      Eugene, Oregon.                                                 *
;*      Copyright  1987       Spectra Physics, Inc.                     *
;*                                                                       *
;*************************************************************************
;*                                                                       *
;*                          HISTORY                                      *
;*                                                                       *
;*      11-March-87 by  Mike Brooks                                     *
;*      Rev 1.00 of code written for "beta" board. No known bugs.       *
;*                                                                       *
;*************************************************************************


;*************************************************************************
;*                                                                       *
;*    REGISTER USEAGE                                                    *
;*                                                                       *
;*    A0 = junk                                                          *
;*    A1 = junk                                                          *
;*    A2 = junk                                                          *
;*    A3 = junk / previous (last) character_frame width                  *
;*    A4 = permanent label buffer character pointer                      *
;*    A5 = permanent IPTR register                                       *
;*                                                                       *
;*    D0 = junk / current character_frame width                          *
;*    D1 = junk                                                          *
;*    D2 = junk                                                          *
;*    D3 = junk                                                          *
;*    D4 = junk                                                          *
;*    D5 = junk                                                          *
;*    D6 = junk                                                          *
;*    D7 = DL7 - loop counter / DH7 - character register                 *
;*                                                                       *
;*************************************************************************
;*                                                                       *
;*    USER CCR REGISTER BITS                                             *
;*                                                                       *
;*    F  E  D  C  B  A  9  8                                             *
;*    |  |  |  |  |  |  |  |_ foreward decode flag                       *
;*    |  |  |  |  |  |  |____ reverse flag                               *
;*    |  |  |  |  |  |_______ data buffer overflow flag                  *
;*    |  |  |  |  |__________ glitch detect info.                        *
;*    |  |  |  |_____________ force use of wand                          *
;*    |  |  |________________ enable P2 addon's                          *
;*    |  |___________________ enable P5 addon's                          *
;*    |______________________                                            *
;*                                                                       *
;*************************************************************************
;*                                                                       *
;*    DECODER MASK REGISTER BITS                                         *
;*                                                                       *
;*    7  6  5  4  3  2  1  0                                             *
;*    |  |  |  |  |  |  |  |_ UPC decoder                                *
;*    |  |  |  |  |  |  |____ EAN decoder                                *
;*    |  |  |  |  |  |_______ UPCD decoder                               *
;*    |  |  |  |  |__________ Code 3 of 9 decoder                        *
;*    |  |  |  |_____________ Interleaved 2 of 5 decoder                 *
;*    |  |  |________________ Code-A-Bar decoder                         *
;*    |  |___________________ Code 93 decoder                            *
;*    |______________________ Code 128 decoder                           *
;*                                                                       *
;*************************************************************************
;*                                                                       *
;*    COMMAND/INFORMATION BYTE SUMMARY                                   *
;*                                                                       *
;*    $3A   UPC A,L segment                                              *
;*    $3F   UPC A,R   "                                                  *
;*    $3E   UPC E     "                                                  *
```

```
;*      $22     P2 ADDON  *                                              *
;*      $25     P5 ADDON  *                                              *
;*      $1D     EAN13,L   *                                              *
;*      $19     EAN8,L    *                                              *
;*      $18     EAN8,R    *                                              *
;*      $07     UPC D     *                                              *
;*      $03     UPC D6    *                                              *
;*      $05     UPC D5    *                                              *
;*      $0A     UPC D4    *                                              *
;*      $09     UPC D3    *                                              *
;*      $06     UPC D2    *                                              *
;*      $0C     UPC D1    *                                              *
;*      $43     C39       label                                          *
;*      $4C     CBAR      *                                              *
;*      $45     I25       *                                              *
;*      $49     C93       *                                              *
;*      $48     C128      *                                              *
;*      $51     Glitch detected                                          *
;*      $52     no such command                                          *
;*      $5C     Start up okay                                            *
;*                                                                       *
;*************************************************************************
;*                                                                       *
;*      MEMORY USEAGE                                                    *
;*                                                                       *
;*      $FFFF - $FC28 ; reserved - internal i/o
;*
;*      PORT LAYOUT
;*
BAUDPT:   EQU       $FC26           ; P19 - timer C latch
SIOMS:    EQU       $FC24           ; P18 - sio mode and sync control
DDR1:     EQU       $FC22           ; P17 - port 1 data direction control
DDR0:     EQU       $FC20           ; P16 - port 0 data direction control
                                    ; P15 - port 0 handshake mode, fast/standard,
                                    ;       bus lock,bud segment bits
                                    ; P14 - timer control and interrupt edge
                                    ;       select
TAHL:     EQU       $FC1A           ; P13 - timer A high latch
TALL:     EQU       $FC18           ; P12 - timer A low latch
TBL:      EQU       $FC16           ; P11 - timer B latch
                                    ; P10 - sio transmit control and status
                                    ; P9  - sio receive control and status
IMASK:    EQU       $FC10           ; P8  - interrupt masks
ILATCH:   EQU       $FC0E           ; P7  - interrupt latches
                                    ; P6  - reserved
                                    ; P5  - reserved
                                    ; P4  - external timer and port 4 i/o
RxDbuf:   EQU       $FC06           ; PH3 - serial receive buffer
TxDbuf:   EQU       $FC05           ; PL3 - serial transmit buffer
                                    ; P2  - reserved
                                    ; P1  - port 1 serial, i/o, interrupt and
                                    ;       bus control bits
                                    ; P0  - port 0 multiplexed address/data lines

;*
;*      INTERNAL 256 BYTES RAM LAYOUT
;*
STACKTOP: EQU       $FBFE           ; top of ram, top of system stack

          ORG       $FB00           ; LOCAL MEMORY DEFINITIONS

LABEL_BUF:          DS.B    33
PARITY:             DS.B    1       ; upc parity bits
DECODER1:           DS.B    1       ; decoder bits register 1
I25LL:              DS.B    1       ; I25 label length
ADDLL:              DS.B    1       ; addon label length (2 or 5)
COUNTER:            DS.B    1       ; timer A latch overflow counter

TALLCNT:            DS.W    1
T1:                 DS.W    1       ; bar/space breakpoint
T2:                 DS.W    1       ; bar/space breakpoint
T3:                 DS.W    1
T4:                 DS.W    1
LAST_WIDTH:         DS.W    1       ; last frame width total
CURRENT_MARGIN:     DS.W    1       ; current working margin
iLDP:               DS.W    1       ; last_decode_point in buffer
iA:                 DS.W    1       ; temporary IPTR storeage
LASTi:              DS.W    1       ; currently found margin by SFS
iPTR:               DS.W    1       ; working data pointer
```

```
;*
;*      EXTERNAL MEMORY LAYOUT
;*
DPTR:       EQU     $A000           ; current data pointer - READ ONLY !!!
                                    ; absolute address $4000
ePTR:       EQU     $8000           ; end of data buffer flag
                                    ; absolute address $6000
CDATA:      EQU     $8000           ; base address of video data buffer
CDEND:      EQU     $A000           ; end+1 of video data buffer
WIDTH:      EQU     CDEND-CDATA     ; width of data array


;*********************************************
;*
;*      CONSTANTS
;*
;*********************************************

REVLVL:     EQU     1               ; software revision level
TINC:       EQU     10000           ; .001 sec clk for data transfer
eINIT:      EQU     CDEND-4         ; end marker for data insertion at start
PBASE:      EQU     $FC00           ; port base address

;*
;* PORT 4 BITS
;*
RESETFE:    EQU     15              ; Reset front end and DPTR output
WDI:        EQU     13              ; Wand data input
GLCH:       EQU     12              ; GLITCH input
OVRFLW:     EQU     11              ; data buffer overflow input
DACK:       EQU     10              ; local bus data acknowledged input
DAVL:       EQU     9               ; local bus data available output

;*
;* USER CCR STATUS BITS
;*
FWD_DECODE: EQU     8               ; upc decoding direction flag
FOREWARD:   EQU     8               ; code 3 of 9 label direction flag
REVERSE:    EQU     9               ; upc seg/parity map reverse flag
OVERFLOW:   EQU     10
GLITCH:     EQU     11              ; tell microcontroller if glitch detected
WAND:       EQU     12              ; allow use of TAI as wand data input
AP2:        EQU     13              ; enable P2 upc addon seg. decoder
AP5:        EQU     14              ; enable P5 upc addon seg. decoder

;*
;* DECODER MASK REGISTER BITS
;*
UPC:        EQU     0               ; upc decoder flag, do it if true
EAN:        EQU     1               ; ean decoder enable flag
UPCD:       EQU     2               ; upc d label decoder enable flag
C39:        EQU     3               ; code 3 of 9 decoder flag, do it if true
I25:        EQU     4               ; interleaved 2 of 5 decoder enable flag
CBAR:       EQU     5               ; code-a-bar decoder enable flag
C93:        EQU     6               ; code 93 decoder enable flag
C128:       EQU     7               ; code 128 decoder enable flag


;****************************************************************
;*                                                              *
;*      START OF CODE                                           *
;*                                                              *
;****************************************************************

            ORG     $0000           ; internal ROM base

;*********************************************
;*
;*      EXCEPTION/INTERRUPT VECTOR TABLE
;*
;*********************************************

VCTTBL:     EQU     *
            DC      IRESET          ;*  COLDSTART
            DC      INMI            ;*  /POWER DOWN
            DC      ISPARE          ;     (ISPARE)
            DC      ISPARE          ;     (IXI2)
```

```
        DC      ISPARE          ;      (ISTRL)
        DC      ITAO            ;* TIMER A OVERFLOW COUNTERS
                                ;  /RESET FRONT END AND DPTR
        DC      ITAI            ;* WAND DATA INPUT
        DC      ISTRN           ;* /OVERFLOW
        DC      ISPARE          ;      (IRSC)
        DC      ISPARE          ;      (IRN)
        DC      ISPARE          ;      (IXI1)
        DC      ISPARE          ;      (ITBO)
        DC      ITBI            ;* /GLITCH
        DC      ISPARE          ;      (IXIO)
        DC      ISPARE          ;      (IXNT)
        DC      ISPARE          ;      (ITC)


;********************************************
;*
;*      STRINGS AND INDEX TABLES
;*
;********************************************

;*                  1234567890123456789012345678901234567890
xf39:      DC.B    '1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZ-. *$/+%'
xr39:      DC.B    'AHGEDJCBIF18754032960.-YX*WV ZKROONTMLSP%+/$'

bar_index: DC.B    00,00,00,07,00,04,10,00,00,02,09,00,06,00,00,00
           DC.B    00,01,08,00,05,00,00,00,03,00,00,00,00,00,00,00

sp_index:  DC.B    00,21,11,00,01,00,00,00,31,00,00,00,00,00,00,00

upc_index: DC.W    $0601,$0000,$0401,$0300,$0900,$0201,$0100,$0501
           DC.W    $0901,$0200,$0101,$0500,$0600,$0001,$0400,$0301

add5tbl:   DC.B    $18,$14,$12,$11,$0C,$06,$03,$0A,$09,$05


;********************************************
;*
;*      EXCEPTION/INTERRUPT HANDLING ROUTINES
;*
;********************************************

;----------POWER ON RESET-----------------------------------------
IRESET:    JMPA     COLDST             ; do a cold start

;----------INMI (POWER DOWN INTERRUPT)----------------------------
INMI:      STOP                        ; go to sleep
           RETI

;----------ISPARE (SPARE INTERRUPT)-------------------------------
ISPARE:    AND      #$0708,P8
           RETI                        ; disable spur. interrupts

;----------IXI2 (EXTERNAL LEVEL 2 INTERRUPT)----------------------

;----------ISTRL (STROBE LOW INTERRUPT)---------------------------

;----------ITAO (TIMER A OVERFLOW COUNTERS)-----------------------
ITAO:      ADD.B    #1,COUNTER
           RETI

;----------ITAI (WAND BASED VIDEO DATA INPUT)---------------------
ITAI:      PUSHM    A0/D0-D2           ; Note that timer A line input is low if
           CLR      D2                 ; scanning a bar
           EXG.B    DH2,COUNTER        ; get timer overflow
           BTST     #13,P4
           JMPR.S   CS,ITAI_4
           TEST.B   #$F8,DH2
           JMPR.S   Z,ITAI_1
           MOVE     #$7FFF,D0          ; TAHL
           JMPR.S   ITAI_2
ITAI_1:    MOVE     P13,D0
           LSR      #4,D0
           ASL.B    #4,DH2
           ADD      D2,D0
ITAI_2:    MOVE     TALLCNT,D1
           MOVE     DPTR,A0
```

```
            MOVEM    D0-D1,(A0)+
            CMP      #GDEND,A0
            JMPR.S   NE,ITAI_3
            MOVE     #GDATA,A0
ITAI_3:     MOVE     A0,DPTR
ITAI_OUT:   POPM     A0/D0-D2
            RETI
ITAI_4:     TEST.B   #$F8,DH2
            JMPR.S   Z,ITAI_5
            MOVE     #$7FFF,D1
            MOVE     D1,TALLCNT
            JMPR.S   ITAI_OUT
ITAI_5:     MOVE     P12,D1
            LSR      #$4,D1
            ASL.B    #4,DH2
            ADD      D2,D1
            MOVE     D1,TALLCNT
            JMPR.S   ITAI_OUT

;----------ISTRH (OVERFLOW)---------------------------------------------
ISTRH:      BCLR     #RESETFE,P4
            BSET     #OVERFLOW,SR
            RETI

;----------IRSC (RECEIVE SPECIAL CONDITION INTERRUPT)-------------------

;----------IRN (RECEIVE DATA NORMAL INTERRUPT)--------------------------

;----------IXI1 (EXTERNAL INTERRUPT #1)---------------------------------

;----------ITBO (TIMER B OUTPUT INTERRUPT)------------------------------

;----------ITBI (GLITCH DETECT INTERRUPT)-------------------------------
ITBI:       BCLR     #RESETFE,P4      ; reset the front end
            BTST     #GLITCH,SR       ; test glitch flag
            JMPR.S   CC,ITBI_1        ; if set, tell controller of glitch
            MOVE.B   #$51,DH7
            CALLA    outch
ITBI_1:     RETI

;----------IXI0 (EXTERNAL INTERRUPT #0)---------------------------------

;----------IXMT (TRANSMITTER INTERRUPT)---------------------------------

;----------ITC (TIMER C INTERRUPT)--------------------------------------


;///////////////////////////////////////////////
;/
;/      SYSTEM SUBROUTINES
;/
;///////////////////////////////////////////////


;*
;* OUTCH send a single character over the 7 lower bits of port 1. Use the
;* output signal DAVL to signal to the HPC that we have placed a character
;* on the "local data bus" and use the input DACK from the HPC to signal
;* acknowledgement on receipt of the character.
;* Enter with the character in DH7.
;* Timing is as follows:
;*                        good data                       good data
;* --> DATA   ???|_____________________|?????????|__________ ...
;*
;* --> DAVL   ____|_____________|______________________|_______ ...
;*
;* <-- DACK   ________|___________|______________________|__ ...
;*             1 2 3 4 5 6   7   8 9 0      1  2   3 4 5 6  7 ...
;*
;*     (1) 68200 monitors *DACK, waiting for it to go high
;*     (2) Then, 68200 places data on PL1/Port D
;*     (3) Next, 68200 asserts *DAVL
;*     (4) Meanwhile, the HPC monitors *DAVL for assertion
;*     (5) Then, HPC reads data off the bus
;*     (6) And, then, HPC asserts *DACK
;*     (7) 68200 looks for *DACK to go low
;*     (8) When 68200 sees it low, it releases *DAVL and exits
;*     (9) HPC monitors *DAVL, when it goes high
```

```
;*      (0) It releases ~DACK and exits
;*
outch:   BTST     #DACK,P4        ; wait for HPC to release ~DACK
         JMPR.S   CC,outch
         MOVE.B   DH7,PL1         ; char ==> PORT 1, bits 0 - 7
         BCLR     #DAVL,P4        ; strobe data into the micro-controller
outch1:  BTST     #DACK,P4        ; loop, waiting for acknowledgement of
         JMPR.S   CS,outch1       ;    receipt of character   part 1
         BSET     #DAVL,P4
         RET

;*
;* Using DACK as an asynchronous serial input get characters in a stream into
;* DL1 - 8 data bits (least significant bit first), 1 start bit, one stop bit.
;* The format is standard.
;*                                                              ________
;*     ‾‾‾‾‾|____|////|////|////|////|////|////|////|////| stop bit
;*           ^         ^data_bit_#0.............data bit #7^
;*        start bit
;*
inch:    MOVE     #TINC/2,P11     ; setup TB to see middle of start char
         CLR      D6
inch1:   BTST     #DACK,P4        ; wait for falling edge of start char
         JMPR.S   CS,inch1
         BSET     #1,P14          ; enable Timer B
         BCLR     #4,P7           ; clear ITBO
inch2:   BTST     #4,P7           ; wait the middle of start char
         JMPR.S   CC,inch2
inch3:   MOVE     #TINC,P11       ; load TB with rate constant
         BCLR     #4,P7           ; clear ITBO flag
inch4:   BTST     #4,P7           ; loop waiting for timeout
         JMPR.S   CC,inch4
         BTST     #DACK,P4        ; then, get current status of DACK
         BEXG     D6,D1           ; exchange it with bits in D1
         ADD      #1,D6           ; low order bit first
         CMP.B    #8,DL6          ; until 8 bits transferred
         JMPA     NE,inch3
         BCLR     #1,P14          ; then, kill TB
inch5:   BTST     #DACK,P4        ; then go look for the stop bit
         JMPR.S   CC,inch5
         RET                      ; return with char in DL1

;*
;* Ti2 and Ti4 increment the address in A5 by 2 or 4 ( one or two element
;*  width ) and tests for over range wraps.
;*
Ti2:     ADD      #2,A5
         CMP      #GDEND,A5       ; A5 < gdata ?
         JMPR.S   LT,Ti21
         MOVE     #GDATA,A5
Ti21:    RET

Ti4:     ADD      #04,A5
         CMP      #GDEND,A5
         JMPR.S   LT,Ti41         ; A5 < gdata ?
         SUB      #WIDTH,A5
Ti41:    RET

;*
;* Ti_2 and Ti_4 decrement the address in A5 by 2 or 4 ( one or two element
;*  width ) and tests for under range wraps.
;*
Ti_2:    SUB      #02,A5
         CMP      #GDATA,A5       ; A5 >= $8000 ?
         JMPR.S   GE,Ti_21
         MOVE     #GDEND-2,A5     ; if A5 < $8000  A5:= BEFE
Ti_21:   RET

Ti_4:    SUB      #04,A5
         CMP      #GDATA,A5       ; A5 >= $8000 ?
         JMPR.S   GE,Ti_41
         ADD      #WIDTH,A5       ; if A5 < $8000   A5:= A5 + WIDTH
Ti_41:   RET

;*
;* WAIT FOR 10 ELEMENTS (MIN) IN DATA BUFFER
;*
room:    MOVE     DPTR,D0         ; get current Dptr
         SUB      IPTR,D0         ; subtract IPTR from it
         JMPR.S   PL,room1        ; get absolute value
```

```
        NEG     D0
room1:  CMP     #20,D0              ; D0 < 20 ? (10 elements)
        JMPR.S  LT,room2            ; yes...keep on looping
        CMP     #WIDTH-20,D0        ; else, test D0 > WIDTH ?
        JMPR.S  LE,room3            ; if not, we're okay. Exit.
room2:  BTST    #OVRFLV,SR          ; else, check the overflow bit
        JMPR.S  CC,room             ; if clear, just keep on looping
                                    ; else, reset system pointers to START
        BCLR    #OVRFLV,SR
        MOVE    #GDATA,D0
        MOVE    D0,DPTR
        ADD     #2,D0
        MOVE    D0,IPTR
        MOVE    D0,ILDP
        MOVE    #eINIT,D0
        MOVE    D0,ePTR
        BTST    #WAND,SR            ; if wand is used for data input skip
        JMPA    CS,room             ; to start
        BSET    #RESETFE,P4
        JMPA    room                ; restart search loop
room3:  RET


;*********************************************************************
;*                                                                   *
;*      START OF MAIN CODE                                           *
;*                                                                   *
;*********************************************************************


;////////////////////////////////////////////
;/
;/      SET UP 68200 MEMORY, PORTS, AND POINTERS
;/
;////////////////////////////////////////////


;*
;* SET UP INITIAL PORTS AND CLEAR LOCAL RAM
;*
COLDST: DI                          ; disable interrupts
        MOVE    #$FBFE,SP           ; initialize the Stack Pointer
        CLR     SIOMS               ; no use for sio - P18
        MOVE    #$F5FF,D0           ; port 1 data direction - P17
        MOVE    D0,DDR1
        CLR     DDR0
        MOVE    #$8840,P15          ; Port 0 handshake/buslock/fast
        MOVE    #$0180,P14          ; Timers and Interrupts control
        CLR     P13
        CLR     P12
        CLR     P10
        CLR     P9
        MOVE    #$0508,P8           ; Interrupt Masks - ACTIVE AT START
                                    ; ITAO - TALL/TAHL overflow
                                    ; ISTRH - data buffer overflow
                                    ; TBI - glitch detect
        MOVE    #$FB00,A0           ; at startup, clear internal RAM
                                    ; FB00 - FBFF is internal RAM
        CLR     D7                  ; D0 = #$0000
COLD1:  MOVE    D7,(A0)+            ; clear memory pointed at by A0
        CMP     #PBASE+4,A0         ; auto-incrementing the pointer
        JMPR.S  NE,COLD1            ; and looping until RAM end
;*
;* SETUP SYSTEM POINTERS
;*
        BCLR    #RESETFE,P4
        MOVE    #GDATA,D0
        MOVE    D0,DPTR
        ADD     #2,D0
        MOVE    D0,IPTR
        MOVE    D0,ILDP
        MOVE    #eINIT,D0
        MOVE    D0,ePTR
;*
;* TELL MICRO-CONTROLLER WE STARTED UP OKAY AND ARE READY FOR
;* STARTUP PARAMETERS
```

```
;*
        MOVE.B  #$5C,DH7
        CALLA   outch
        MOVE.B  #REVLVL,DH7
        CALLA   outch
;*
;* LOOP WAITING FOR SETUP PARAMETERS
;*
;* Begin setups for mode, decoder selection, interleaved 2 of 5 label length,
;* etc. This argument expects a data packet in the form:
;*        byte #1          byte #2         byte #3
;*        CCR mask         DECODER1 mask   I25LL
;* This routines installs the parameters in place and returns them, ordered,
;* to the HPC microcontroller.
;*
SETUP:  CALLA   inch            ; get CCR mask
        MOVE.B  DL1,DH7
        CALLA   inch            ; get DECODER mask
        MOVE.B  DL1,DECODER1
        CALLA   inch            ; get I25 label length
        MOVE.B  DL1,I25LL
        MOVE    D7,SR           ; apply the CCR mask and return it to
        CALLA   outch           ; the HPC to see if we got it right
        MOVE.B  DECODER1,DH7    ; return DECODER MASK it to the HPC
        CALLA   outch
        MOVE.B  I25LL,DH7       ; return I25LL to the HPC
        CALLA   outch
;*
        CLR     D0              ; setup addon label length
        BTST    #AP5,SR
        JMPR.S  CC,SA1
        MOVE.B  #5,DL0
        JMPR.S  SA2
SA1:    BTST    #AP2,SR
        JMPR.S  CC,SA2
        MOVE.B  #2,DL0
SA2:    MOVE.B  DL0,ADDLL
;*
        BTST    #WAND,SR        ; check for wand input
        JMPR.S  CC,START1       ; if required
        BSET    #6,P14          ; turn on timer A
        BSET    #9,P8           ; enable the interrupt
        BSET    #10,P8          ; and the overflow irq
        JMPR.S  START2
;*
START1: BSET    #RESETFE,P4
START2: EI


;*
;* SEARCH FOR START - LOCATE A POSSIBLE LABEL START
;*
sfs:    CALLA   room
;*
;* FIND A LARGE WHITE SPACE SUCH THAT e0 > e1+e2+e3
;*
margin: MOVE    IPTR,A5
        MOVE    (A5),D0         ; get e0
        CALLA   TI2
        MOVE    (A5),D1         ; get e1
        CALLA   TI2             ; point at e2
        ADD     (A5),D1         ; D1 = e1+e2
        CALLA   TI2             ; point at e3
        ADD     (A5),D1         ; D1 = e1+e2+e3
        CMP     D0,D1           ; e1+e2+e3 < e0 ?
        JMPR.S  LT,sfs_out      ; if so, we're okay. get out
        MOVE    IPTR,A5         ; else, move IPTR to next space
        ADD     #04,A5          ; and restart test
        CMP     #CDEND,A5
        JMPR.S  LT,margin1
        SUB     #WIDTH,A5
margin1: MOVE   A5,IPTR
        JMPR.S  sfs
;*                              ; we've found a wide space
sfs_out: MOVE   IPTR,D2         ; update the Eptr
        MOVE    D2,LAST1        ; store away IPTR address
        SUB     #$92,D2         ; subtract 146 from IPTR ADDress
```

```
         CMP      #GDATA,D2          ; D2 > gdend  ?
         JMPR.S   GE,sfs_out1        ; yes! exit
         ADD      #WIDTH,D2          ; not wrap pointer
sfs_out1: MOVE    D2,ePTR
;*
;* START THE CODE 3 OF 9 DECODER
;*
code39:  BTST     #C39,DECODER1      ; test switch, do code 3 of 9 ?
         JMPA     CC,codeUPC         ; switch not closed, skip it
         CALLA    T12                ; point at e4
         ADD      (A5),D1            ; D1 = e1+e2+e3+e4
         CMP      D0,D1              ; e1+e2+e3+e4 > e0  ?
         JMPA     GT,codeUPC         ; if so, go to code UPC
;*
         CLR      LABEL_BUF          ; init label string buffer and pointer
         MOVE     #LABEL_BUF+1,A4
;*
         CALLA    c311a              ; LOOK FOR A START CHARACTER
         CMP.B    #$0C,DL6           ; bar_pattern = %01100 ?
         JMPR.S   NE,c39_1           ; no...try foreward start
         CMP.B    #01,DL5            ; space_pattern = %0001
         JMPA     NE,codeUPC         ; not a match. Quit.
         BCLR     #FOREWARD,SR       ; Yes! we are decoding label reversed !!
         JMPR.S   c317a
c39_1:   CMP.B    #06,DL6            ; or, if bar = %00110 & space = %1000
         JMPA     NE,codeUPC         ; no start pattern found. Quit!
         CMP.B    #08,DL5
         JMPA     NE,codeUPC
         BSET     #FOREWARD,SR       ; we are decoding foreward
;*
;* NOW, TEST CHAR WIDTHS PER SPEC. 4.3.17 - 4.3.23
;*
c317a:   CALLA    c317
                                     ; AT EXIT D2 = e1+e2+e3+e4+e5+e6+e7+e8+e9
                                     ;         D1 = narrowest space
                                     ;         D0 = narrowest bar
c318a:   MOVE     D1,D3              ; get narrowest space
         LSR      #1,D3              ; calc NS*1.5
         ADD      D1,D3
         CMP      T2,D3              ; NS*1.5 > widest_space ?
         JMPA     GT,codeUPC         ; yes, quit. Less than min elem ratio.
         MOVE     D1,D3
         ASL      #2,D3              ; calc NS*5.0
         ADD      D1,D3
         CMP      T2,D3              ; NS*5.0 < wisest_space ?
         JMPA     LT,codeUPC         ; yes, quit. greater than max elem ratio
c319a:   MOVE     D0,D3              ; get narrowest bar
         ASL      #2,D3              ; calc NB*5.0
         ADD      D0,D3
         CMP      T1,D3              ; NB*5.0 < widest bar ?
         JMPA     LT,codeUPC         ; yes, quit. greater than max elem ratio
c320a:   MOVE     D0,D3              ; else
         LSR      #1,D3              ; calc NB*1.5
         ADD      D0,D3
         CMP      T1,D3              ; NB*1.5 > widest_bar
         JMPA     GT,codeUPC
c321a:   MOVE     D1,D3              ; get the narrowest space
         ADD      D3,D3              ; calc NS*3
         ADD      D1,D3
         CMP      D3,D0              ; narrowest_bar > NS*3 ?
         JMPA     GT,codeUPC
c322a:   MOVE     D0,D3              ; get the narrowest bar
         ADD      D3,D3              ; calc NB*3
         ADD      D0,D3
         CMP      D3,D1              ; narrowest_space > NB*3
         JMPA     GT,codeUPC         ; EXIT WITH  DH7 ==> CHARACTER
                                     ;            D4  ==> BAR PATTERN
                                     ;            D2  ==> CURRENT CHAR_WIDTH
                                     ;            D1  ==> NARROWEST SPACE
                                     ;            D0  ==> NARROWEST BAR
         MOVE     D2,A3              ; store last_width at start
;*
;* START A LOOP LOOKING FOR CHARACTERS
;*
c39loop: MOVE     iPTR,A5            ; move iPTR to next character
         ADD      #20,A5
         CMP      #GDEND,A5
         JMPR.S   LT,c39loop1
         SUB      #WIDTH,A5
c39loop1: MOVE    A5,iPTR
```

```
        CALLA     room            ; loop, if necessary, for data
        CALLA     c311a           ; GET BAR AND SPACE PATTERNS
;*
;* Use the bar and space patterns calculated above to get a character. If the
;* character or index is invalid exit the decoder. The following code imple-
;* ments steps 4.3.14, 4.3.15, and 4.3.16 inthe technical specification.
;*
;* If bar pattern = 0, then if space pattern = 7  chrptr = 44
;*                          if sapce pattern = 11 chrptr = 43
;*                          if space pattern = 13 chrptr = 42
;*                          if space pattern = 14 chrptr = 41
;*                          if not these inveld pattern, quit.
;* otherwise, calculate chrptr as:
;*    [10*(space_index(space_pattern)-1)] + (bar_index(bar_pattern))
;* Then, using chrptr as an index use either table xf39 or xr39 to get
;* a character.
;*
c314:   CMP.B     #0,DL4          ; bar pattern = 0 ?
        JMPR.S    NE,c315         ; if yes, and
        CMP.B     #07,DL5         ; space pattern = 7
        JMPR.S    NE,c314_1
        MOVE.B    #44,DL0         ; then, chrptr = 44
        JMPR.S    c316
c314_1: CMP.B     #11,DL5         ; if space pattern = 11
        JMPR.S    NE,c314_2
        MOVE.B    #43,DL0         ; then, chrptr = 43
        JMPR.S    c316
c314_2: CMP.B     #13,DL5         ; if space pattern = 13
        JMPR.S    NE,c314_3
        MOVE.B    #42,DL0         ; then, chrptr = 42
        JMPR.S    c316
c314_3: CMP.B     #14,DL5         ; if space pattern = 14
        JMPA      NE,codeUPC      ; else, invalid pattern. Quit.
        MOVE.B    #41,DL0         ; then, chrptr = 41
        JMPR.S    c316
c315:   MOVE      #sp_index,A0    ; get space index table base
        ADD       D5,A0           ; calc offet
        MOVE.B    (A0),DL0        ; get (space_index(space_pattern))
        JMPA      EQ,codeUPC      ; if = 0, invalid index. Quit.
        SUB.B     #1,DL0
        MOVE      #bar_index,A0   ; get bar index table base
        ADD       D4,A0           ; calc bar table offset
        MOVE.B    (A0),DL1        ; get (bar_index(bar_pattern))
        JMPA      EQ,codeUPC      ; if = 0, invalid index. Quit.
        ADD.B     DL1,DL0         ; if okay, calc chrptr
c316:   CLR.B     DH0
        BTST      #FOREWARD,SR    ; decide char table to use
        JMPR.S    CS,c316_1
        MOVE      #xr39,A0        ; if foreward = false, use reverse table
        JMPR.S    c316_2
c316_1: MOVE      #xf39,A0        ; if foreward = true, use foreward table
c316_2: ADD       D0,A0
        MOVE.B    (A0),DH7        ; get character
                                  ; EXIT WITH  D5  ==> SPACE PATTERN
                                  ;            D4  ==> BAR PATTERN
                                  ;            DH7 ==> CHAR
;*...
;* This subroutine is called from the main flow of the algorithm. It tests
;* the sizes of elements within a character for correct width ratios. It
;* uses the values of the widest bar and space, stored respectively in T1
;* and T2 by the argument c311a previously executed. Now, find the total
;* of the elements making up the current character and find the narrowest
;* bar and space. The sections ofm this argument correspond to sections
;* 4.3.17 through 4.3.23 in the technical documentation.
;*
        CALLA     c317            ; AT EXIT D2 = char width
                                  ;         D1 = narrowest space
                                  ;         D0 = narrowest bar
c318:   MOVE      D1,D3           ; get narrowest space
        LSR       #1,D3           ; calc NS*1.5
        ADD       D1,D3
        CMP       T2,D3           ; NS*1.5 > widest_space ?
        JMPA      GT,codeUPC      ; yes, quit. less than min elem ratio
        MOVE      D1,D3
        ASL       #2,D3           ; calc NS*5.0
        ADD       D1,D3
        CMP       T2,D3           ; NS*5.0 < wisest_space ?
        JMPA      LT,codeUPC      ; yes, quit. greater than max elem ratio
c319:   MOVE      D0,D3           ; get narrowest bar
        ASL       #2,D3           ; calc NB*5.0
```

```
        ADD     D0,D3
        CMP     T1,D3           ; NS*5.0 < widest bar ?
        JMPA    LT,codeUPC      ; yes, quit. greater than max elem ratio
c320:   CMP.B   #0,DL4          ; if bar pattern = 0 skip this step
        JMPR.S  EQ,c321
        MOVE    D0,D3           ; else
        LSR     #1,D3           ; calc NB*1.5
        ADD     D0,D3
        CMP     T1,D3           ; NB*1.5 > widest_bar
        JMPA    GT,codeUPC
c321:   MOVE    D1,D3           ; get the narrowest space
        ADD     D3,D3           ; calc NS*3
        ADD     D1,D3
        CMP     D3,D0           ; narrowest_bar > NS*3 ?
        JMPA    GT,codeUPC
c322:   MOVE    D0,D3           ; get the narrowest bar
        ADD     D3,D3           ; calc NB*3
        ADD     D0,D3
        CMP     D3,D1           ; narrowest_space > NB*3
        JMPA    GT,codeUPC      ;       DH7 ==> CHARACTER
                                ;       D4  ==> BAR PATTERN
                                ;       D2  ==> CURRENT CHAR_WIDTH
                                ;       D1  ==> NARROWEST SPACE
                                ;       D0  ==> NARROWEST BAR
;*
;* Compute the ratio of the sum of the elements in the current character to
;* element e0. If this sum is greater than max char ratio or less than min
;* char ratio quit.  Reference section 4.3.7 of technical document.
;*
c307:   MOVE    A3,D4           ; get a copy of last char width
        LSR     #2,D4           ; LW/4
        ADD     A3,D4           ; LW/4 + LW = LW*1.25
        CMP     D4,D2           ; CW > LW*5/4
        JMPA    GT,codeUPC
        MOVE    D2,D4           ; D4 = CW
        LSR     #2,D4           ; D4 = CW/4
        ADD     D2,D4
        CMP     D4,A3           ; LW < CW*5/4
        JMPA    LT,codeUPC
;*
;* Compute the ratio of the sum of the elements in the current character to
;* element e0. If this ratio is greater than max gap ratio or less than min
;* gap ratio quit decoder. Reference section 4.3.8 of technical document.
c308:   MOVE    IPTR,A5         ; get current IPTR
        MOVE    (A5),D4         ; get e0
        ADD     D4,D4           ; e0*2
        CMP     D4,D2           ; CW < 2*e0 ?
        JMPA    LT,codeUPC      ; yes. Quit.
        MOVE    D4,D5           ; save e0*2
        ASL     #4,D4           ; e*32
        SUB     D5,D4           ; e*30
        CMP     D4,D2           ; char_width > 30*e0 ?
        JMPA    HI,codeUPC      ; yes. Quit.
        MOVE    D2,A3           ; else, last_width := char_width
;*
;* TEST CHARACTER FOUND FOR STOP CHARACTER
;*
        CMP.B   #42,DH7         ; if 042d ( "*" ) = stop char
        JMPR.S  EQ,exitloop     ; exit
        MOVE.B  DH7,(A4)+       ; else, store char in label buffer
        ADD.B   #1,LABEL_BUF    ; update buffer char count
        MOVE.B  LABEL_BUF,DL0   ; test label buffer for too many char's
        CMP.B   #32,DL0         ; if D0 <= 32 keep on going
        JMPA    LE,c39loop      ; else, too many char's. Quit.
        JMPA    codeUPC
exitloop: MOVE  IPTR,A5         ; look for a trailing margin
        ADD     #20,A5          ; move pointer to trailing margin
        CMP     #CDEND,A5       ; ref 4.3.10
        JMPR.S  LT,c39_3
        SUB     #WIDTH,A5
c39_3:  MOVE    (A5),D0         ; get possible margin (e10)
        MOVE.B  #04,DL7
        CLR     D1
c39_4:  CALLA   T1_2            ; move pointer to e9...e6
        ADD     (A5),D1         ; sum e6+e7+e8+e9
        DJNZ.B  DL7,c39_4
        CMP     D0,D1           ; e6+e7+e8+e9 > e10 ?
        JMPA    GT,codeUPC      ; yes, margin too small. Quit
                                ; else, LABEL FOUND !!!
        MOVE.B  #$43,DH7        ; sign the label type code 3 of 9
```

```
            CALLA     outch
            MOVE      LABEL_BUF,A0       ; PRINT THE DECODED LABEL STRING
            MOVE.B    (A0)+,DL7
            MOVE.B     DL7,DH7
            CALLA     outch
            BTST      #FOREWARD,SR
            JMPR.S    CC,c39_6           ; if forward = false, reverse label
c39_5:      MOVE.B    (A0)+,DH7
            CALLA     outch
            DJNZ.B    DL7,c39_5
            JMPA      found_label        ; then finish up
c39_6:      MOVE.B    -(A4),DH7          ; get char n,n-1,n-2,...3,2,1
            CALLA     outch              ; send them out
            DJNZ.B    DL7,c39_6          ; until done
            JMPA      found_label
;*
;* The following routine is the equivalent of the routines 4.3.11,
;* 4.3.12 and 4.3.13 found in the technical documentation. The bar
;* and space breakpoints are found by multiplying the largest element
;* amoung e1, e3, e5, e7, and e9 by 0.700. Similarly, the largest space
;* amoung elements e2, e4, e6, and e8 is multiplied by 0.700.
;*
;* The results are used to generate two binary character patterns. A
;* register is set to zero and each bar (or space in the case of the
;* space pattern) is compared with the wide/narrow breakpoint. If the
;* element is larger than the breakpoint the register is incremented by
;* one, then the register is multiplied by two. In the case of the bar
;* pattern, if the result equals 31 (all wide bars) the register is set
;* to zero.
;*
;* The argument returns with bar pattern in D4 and space pattern in D5
;* and the largest bar in T1 and the largest space in T2.
;*
c311a:      CLR       D4                 ; clear storeage for bar pattern
            CLR       D5                 ; clear storeage for space pattern
            MOVE      IPTR,A5            ; get current IPTR
            CALLA     T12                ; move pointer to e1
            MOVE      (A5),D0            ; and get e1
            MOVE.B    #4,DL7             ; set up a loop counter
c311a_1:    CALLA     T14                ; move pointer to next bar
            CMP       (A5),D0            ; if D0 >= eN  leave old value in D0
            JMPR.S    GE,c311a_2
            MOVE      (A5),D0            ; else, if D0 < eN , D0 := eN
c311a_2:    DJNZ.B    DL7,c311a_1        ; decrment the loop counter
            MOVE      D0,T1              ; store away the largest bar found
            MOVE      D0,D1              ; multiply D0 by 11/16 == .69
            ADD       D1,D1              ; n*2
            ADD       D1,D0              ; n*3
            ASL       #2,D1              ; n*8
            ADD       D1,D0              ; n*3 + n*8 = n*11
            LSR       #4,D0              ; n*11/16
            MOVE.B    #5,DL7             ; get the bar pattern
            MOVE      IPTR,A5            ; get IPTR address
            CALLA     T12                ; point at e1
c312:       ADD       D4,D4              ; D4*2    rem: D4 is bar_pattern
            CMP       (A5),D0            ; bar_bkpt > eN ?
            JMPR.S    GT,c312_1          ; no, increment D4 by 1
            ADD       #1,D4
c312_1:     CALLA     T14                ; move pointer to next bar
            DJNZ.B    DL7,c312           ; decrement the loop counter
            CMP.B     #31,DL4            ; then, test if bar_pattern = 31
            JMPR.S    NE,c311b
            CLR       D4                 ; if D4 = 31 , then D4 = 0
;*
c311b:      MOVE      IPTR,A5            ; FIND THE LARGEST SPACE
            CALLA     T14
            MOVE      (A5),D0
            MOVE.B    #3,DL7             ; set up a loop count
c311b_1:    CALLA     T14                ; increment pointer to next space
            CMP       (A5),D0            ; searching for the largest space
            JMPR.S    GE,c311b_2         ; D0 < eN ?
            MOVE      (A5),D0            ; no, replace old eN with new eN
c311b_2:    DJNZ.B    DL7,c311b_1
            MOVE      D0,T2              ; store away the largest space
            MOVE      D0,D1              ; CALC SPACE BREAKPOINT
            ADD       D1,D1
            ADD       D1,D0              ; n*11/16 == .69
            ASL       #2,D1
            ADD       D1,D0
            LSR       #4,D0
```

```
                                        ; CALC SPACE PATTERN
          MOVE      IPTR,A5             ; get back IPTR
          MOVE.B    #4,DL7              ; set up loop counter
c313:     CALLA     T14
          ADD       D5,D5               ; D5*2
          CMP       (A5),D0             ; space_bkpt > eN  ?
          JMPR.S    GT,c313_1           ; no, increment D5 by 1
          ADD       #1,D5
c313_1:   DJNZ.B    DL7,c313            ; EXIT WITH D5 ==> SPACE PATTERN
          RET                           ;           D4 ==> BAR PATTERN
                                        ;           T1 ==> LARGEST BAR
                                        ;           T2 ==> LARGEST SPACE

                                        ; FIND THE NARROWEST BAR & SPACE AND TOTALS
c317:     MOVE      IPTR,A5             ; get current IPTR
          CALLA     T12                 ; point at element e1, first bar
          MOVE      (A5),D0             ; get e1
          MOVE      D0,D2               ; ADD e1 to current char_width total
          CALLA     T12                 ; move to the first space
          MOVE      (A5),D1             ; and get it
          ADD       D1,D2
          MOVE.B    #7,DL7              ; set up a loop counter
c317_1:   CALLA     T12                 ; next bar
          CMP       (A5),D0             ; D0 <= eN
          JMPR.S    LE,c317_2           ; no
          MOVE      (A5),D0             ; yes, replace old eN in D0
c317_2:   ADD       (A5),D2             ; sum all bars into total
          SUB.B     #1,DL7
          JMPR.S    EQ,c317_4
          CALLA     T12                 ; next space
          CMP       (A5),D1             ; D1 <= eN
          JMPR.S    LE,c317_3           ; no
          MOVE      (A5),D1             ; yes, replaced D0 with eN
c317_3:   ADD       (A5),D2             ; add spaces to total
          DJNZ.B    DL7,c317_1          ; AT EXIT D2 = e1+e2+e3+e4+e5+e6+e7+e8+e9
c317_4:   RET                           ;         D1 = narrowest space
                                        ;         D0 = narrowest bar


;*
;* START THE UPC/EAN DECODER
;*
          INCLUDE   UPC

;*
;* IF A GOOD LABEL IS FOUND, RESET SYSTEM POINTERS AND RE-ENTER THE DECODING
;* LOOP.
;*
found_label: BCLR   #RESETFE,P4
          MOVE      #eINIT,D0           ; reset system pointers
          MOVE      D0,ePTR
          MOVE      #GDATA,D0
          MOVE      D0,DPTR
          ADD       #2,DPTR
          MOVE      D0,IPTR
          MOVE      D0,ILDP
          BTST      #WAND,SR
          JMPA      CC,sfs
          BSET      #RESETFE,P4         ; then, start the decoders
          JMPA      sfs
;*
;* IF NO GOOD LABEL FOUND, INCREMENT IPTR TO NEXT SPACE AND RESTART
;* THE DECODERS.
;*
no_decode: MOVE     LASTI,A5            ; else, move IPTR to next space
          ADD       #04,A5              ; and restart test
          CMP       #GDEND,A5
          JMPR.S    LT,no_d1
          SUB       #WIDTH,A5
no_d1:    MOVE      A5,IPTR
          JMPA      sfs                 ; loop back to top of argument

          END

;**********************************************************************
;*                                                                    *
;*                          END OF CODE                               *
;*                                                                    *
;**********************************************************************
```

```
;//////////////////////////////////////////////////////////////////////
;/                                                                    /
;/ UPC/EAN DECODER.                                                   /
;/                                                                    /
;/ FIXED REGISTER ASSIGNMENTS:                                        /
;/                  A5 = base address of latch                        /
;/                  A6 = base pointer (always $00000000)              /
;/                                                                    /
;//////////////////////////////////////////////////////////////////////

;* GET A CHARACTER. Sum elements e0 to e3 to calculate the current_char_width.
;* If e0 is a bar (bit 1 of IPTR is clear), then set T1 := e0 + e1 and T2 :=
;* e1 + e2. If e0 is a space (bit 1 of IPTR is set), then set T1 := e2 + e3
;* and T2 := e1 + e2. Set ratio1 := T1/char_width and ratio2 := T2/char_width.
;* Then, calculate character weight as:
;*      - if ratio1 < thresh1, weight := 0
;*      - if ratio1 < thresh2, weight := 4
;*      - if ratio1 < thresh3, weight := 8
;*                else    , weight := 12
;*                                                   then:
;*      - if ratio2 < thresh1, weight := weight+0
;*      - if ratio2 < thresh2, weight := weight+1
;*      - if ratio2 < thresh3, weight := weight+2
;*                else    , weight := weight+3
;*
c921:       MOVE     #upc_index,A0    ; get character table base
                                      ; weight*2 is char offset base
            MOVE     IPTR,A5          ; ref 9.3.21
            MOVE     (A5),D0          ; e0
            CALLA    T12
            ADD      (A5),D0          ; e0+e1
            CALLA    T12
            ADD      (A5),D0          ; e0+e1+e2
            CALLA    T12
            ADD      (A5),D0          ; D0 := e0+e1+e2+e3
;*
            BTST     #1,IPTR          ; is e0 a bar or a space ?
            JMPR.S   CC,bar_921       ; if bit 1 is set e0 is a SPACE !!
sp_921:     MOVE     (A5),D2          ; e3
            CALLA    T1_2
            MOVE     (A5),D4          ; e2
            ADD      D4,D2            ; D2 = T1 := e3+e2
            CALLA    T1_2
            ADD      (A5),D4          ; D4 = T2 := e2+e1
            JMPR.S   c921_1
bar_921:    CALLA    T1_2
            MOVE     (A5),D4          ; e2
            CALLA    T1_2
            MOVE     (A5),D2          ; e1
            ADD      D2,D4            ; D4 = T2 := e1+e2
            CALLA    T1_2
            ADD      (A5),D2          ; D2 = T1 := e0+e1
;*
;*                                      D0 = CHAR_WIDTH
;*                                      D2 = T1
;*                                      D4 = T2
c921_1:     MOVE     #70,A2           ; T1
            MULU     A2,D2
            DIVU     D0,D2            ; D2 = T1*70/CW
            CMP      #25,D2
                                      ; D2 < 25 , weight = 0
            JMPR.S   LE,c921_5
c921_2:     CMP      #35,D2
            JMPR.S   GT,c921_3        ; D2 < 35 , weight = 4
            ADD      #8,A0
            JMPR.S   c921_5
c921_3:     CMP      #45,D2
            JMPR.S   GT,c921_4        ; D2 < 45 , weight = 8
            ADD      #16,A0
            JMPR.S   c921_5
c921_4:     ADD      #24,A0           ; D2 > 45 , weight = 12
c921_5:     MULU     A2,D4            ; T2
            DIVU     D0,D4            ; D4 = T2*70/CW
            CMP      #25,D4
            JMPR.S   LE,c922          ; D4 < 25 , weight := weight+0
c921_6:     CMP      #35,D4
```

```
          JMPR.L    c921_7            ; D4 < 35 , weight := weight+1
          ADD       #2,A0
          JMPR.S    c922
c921_7:   CMP       #45,D4
          JMPR.S    GT,c921_8         ; D4 < 45 , weight := weight+2
          ADD       #4,A0
          JMPR.S    c922
c921_8:   ADD       #6,A0             ; D4 > 45 , weight := weight+3
                                      ; AT EXIT    A0 ==> WEIGHT+BASE
                                      ;            D0 ==> CHAR_WIDTH
;*
;* USE THE CALCULATED WEIGHT AS AN OFFSET INTO THE CHARACTER LOOKUP TABLE.
;*                                    ; REF 9.3.22
c922:     MOVE      (A0),D1           ; AT EXIT    DH1 ==> CHARACTER
                                      ;            DL1 ==> PARITY
                                      ;            D0  ==> CHAR_WIDTH
;*
;* CHECK FOR AMBIGUITIES. (REF 9.3.23)  If character is a "1" or a "2" test
;* character in accordance with the ambiguity rules summarized in the follow-
;* ing table:
;*       If  CHAR   and  e0     and  PARITY  and  INEQUALITY   then  CHAR BECOMES
;*  (1)      "2"         space       even         e2 > e1            "8", else "2"
;*  (2)      "2"         bar         even         e2 < e1            "8", else "2"
;*  (3)      "2"         space       odd          e2 < e1            "8", else "2"
;*  (4)      "2"         bar         odd          e2 > e1            "8", else "2"
;*  (5)      "1"         space       odd          e3 > e2            "7", else "1"
;*  (6)      "1"         bar         odd          e0 > e1            "7", else "1"
;*  (7)      "1"         space       even         e2*2/3 > e3        "7", else "1"
;*  (8)      "1"         bar         even         e1*2/3 > e0        "7", else "1"
;*
c923:     CMP.B     #$02,DH1          ; If character = "2"
          JMPR.S    NE,c923_5
          MOVE      IPTR,A5           ; get IPTR
          CALLA     T12
          MOVE      (A5),D3           ; get D3 = e1
          CALLA     T12               ; and
          MOVE      (A5),D4           ; get D4 = e2
          BTST      #1,A5             ; and e0 = SPACE (bit 1 set )
          JMPR.S    CC,c923_2
          CMP.B     #0,DL1            ; and PARITY EVEN (= 1)
          JMPR.S    EQ,c923_1
          CMP       D3,D4             ; and e2 > e1
          JMPR.S    LE,c923_4
          MOVE.B    #$08,DH1          ; then char = "8", else "2"
          RET
c923_1:   CMP       D3,D4             ; If char = "2" and e0 = SPACE and PARITY
          JMPR.S    GE,c923_4         ; = ODD and e2 < e1
          MOVE.B    #$08,DH1          ; then char = "8", else "2"
          RET
c923_2:   CMP.B     #0,DL1            ; If char = "2" and e0 = BAR
          JMPR.S    EQ,c923_3         ; and PARITY = EVEN
          CMP       D3,D4             ; and e2 < e1
          JMPR.S    GE,c923_4
          MOVE.B    #$08,DH1          ; then char = "8", else "2"
          RET
c923_3:   CMP       D3,D4             ; If char = "2" and e0 = BAR and PARITY
          JMPR.S    LE,c923_4         ; = ODD and e2 > e1
          MOVE.B    #$08,DH1          ; then char = "8", else "2"
c923_4:   RET
c923_5:   CMP.B     #$01,DH1          ; If character not = "1" or "2", quit.
          JMPR.S    NE,c923_4         ; If character = "1"
          MOVE      IPTR,A5           ; get IPTR
          BTST      #1,A5             ; e0 = SPACE ?
          JMPR.S    CC,c923_7
          CALLA     T14
          MOVE      (A5),D3           ; get e2
          CALLA     T12
          MOVE      (A5),D4           ; get e3
          CMP.B     #0,DL1            ; and PARITY = EVEN
          JMPR.S    EQ,c923_6
          ADD       D3,D3             ; D3 = e2*2
          ADD       D4,D4
          ADD       (A5),D4           ; D4 = e3*3
          CMP       D4,D3             ; 2*e2 > 3*e3      (e2*2/3 > e3)
          JMPR.S    LE,c923_4
          MOVE.B    #$07,DH1
          RET
c923_6:   CMP       D3,D4             ; PARITY = ODD
          JMPR.S    LE,c923_8         ; e3 > e2
          MOVE.B    #$07,DH1
```

```
            RET
c923_7:     MOVE    (A5),D3         ; get e0
            CMP.B   #0,DL1          ; BAR
            JMPR.S  EQ,c923_9       ; PARITY set = EVEN
            ADD     D3,D3
            ADD     (A5),D3         ; D3 = e0*3
            CALLA   T12
            MOVE    (A5),D4
            ADD     D4,D4           ; D4 = e1*2
            CMP     D3,D4           ; e1*2 > e0*3      (e1*2/3 > e0)
            JMPR.S  LE,c923_8
            MOVE.B  #a07,DH1
c923_8:     RET
c923_9:     CALLA   T12             ; PARITY clear = ODD
            MOVE    (A5),D4         ; D4 = e1
            CMP     D4,D3           ; e0 > e1
            JMPR.S  LE,c923_8
            MOVE.B  #a07,DH1        ; AT EXIT   DH1 ==> CHARACTER
            RET                     ;           DL1 ==> PARITY BIT
                                    ;           D0  ==> CHAR_WIDTH
;*
;* CHECK WIDTHS (REF. 9.3.24). If current_char_width/last_char_width > max_
;* char_ratio then return with char too big indication. If ratio < 1/max_
;* char_ratio then return with char too small indication. Otherwise, return
;* with successful indication. (max_char_ratio = 5/4).
;*
c924:       CLR     D6              ; TEST FOR TOO BIG
            MOVE    A3,D5           ; get last width
            LSR     #2,D5
            ADD     A3,D5           ; LW*5/4
            CMP     D5,D0           ; CW > LW*5/4
            JMPR.S  LE,c924_1       ; if okay, try test for too small
            MOVE.B  #$FF,DH6        ; too big indicator
            RET
c924_1:     MOVE    D0,D5           ; TEST FOR TOO SMALL
            LSR     #2,D5
            ADD     D0,D5           ; CW*5/4
            CMP     D5,A3           ; LW > CW*5/4 - REM A3 = last_width
            JMPR.S  LE,c924_2
            MOVE.B  #$FF,DL6        ; AT EXIT   DH1 ==> CHARACTER
c924_2:     RET                     ;           DL1 ==> PARITY BIT
                                    ;           D0  ==> CHAR_WIDTH
                                    ;           D6  ==> 0000 = ok
                                    ;                   00FF = too small
                                    ;                   FF00 = too large


;*
;* DECODE PARITY SEGMENT MAP. If reverse is true, then reverse the order of
;* the PARITY bits. If segment string length is 6 use the following table to
;* look up the segment type.
;*    00/3A   oooooo    segment type, UPC A,L     encoded digit = 0
;*    07      oooeee    segment type, UPC D       encoded digit = 0
;*    0B/1D   ooeoee    segment type, EAN13,L     encoded digit = 1
;*    0D/1D   ooeeoe    segment type, EAN13,L     encoded digit = 2
;*    0E/1D   ooeeeo    segment type, EAN13,L     encoded digit = 3
;*    13/1D   oeooee    segment type, EAN13,L     encoded digit = 4
;*    15/1D   oeoeoe    segment type, EAN13,L     encoded digit = 7
;*    16/1D   oeoeeo    segment type, EAN13,L     encoded digit = 8
;*    19/1D   oeeooe    segment type, EAN13,L     encoded digit = 5
;*    1A/1D   oeeoeo    segment type, EAN13,L     encoded digit = 9
;*    1C/1D   oeeeoo    segment type, EAN13,L     encoded digit = 6
;*    23/3E   eoooee    segment type, UPC E       encoded digit = 6
;*    25/3E   eooeoe    segment type, UPC E       encoded digit = 9
;*    26/3E   eooeeo    segment type, UPC E       encoded digit = 5
;*    29/3E   eoeooe    segment type, UPC E       encoded digit = 8
;*    2A/3E   eoeoeo    segment type, UPC E       encoded digit = 7
;*    2C/3E   eoeeoo    segment type, UPC E       encoded digit = 4
;*    31/3E   eeoooe    segment type, UPC E       encoded digit = 3
;*    32/3E   eeooeo    segment type, UPC E       encoded digit = 2
;*    34/3E   eeoeoo    segment type, UPC E       encoded digit = 1
;*    38/3E   eeeooo    segment type, UPC E       encoded digit = 0
;*    3F      eeeeee    segment type, UPC A,R     encoded digit = 0
;*
;* If the length of the segment string is 4, use the next table to look up
;* the segment type.
;*    00/19   oooo    segment type, EAN8,L    encoded digit = 0
;*    0F/18   eeee    segment type, EAN8,L    encoded digit = 0
;*    03      ooee    segment type, UPC D6    encoded digit = 6
;*    05      oeoe    segment type, UPC D5    encoded digit = 5
```

```
;*      06      o••o    segment type, UPC D2      encoded digit = 2
;*      09      •oo•    segment type, UPC D3      encoded digit = 3
;*-     0A      •o•o    segment type, UPC D4      encoded digit = 4
;*      0C      ••oo    segment type, UPC D1      encoded digit = 1
;*
;* Else, the segment string is an addon or an error.
;*
;*      22              segment type, 2 char addon
;*      25              segment type, 5 char addon
;*      00              segment type, ERROR!!!
;*
c925:       MOVE.B   PARITY,DL3         ; get PARITY pattern
            MOVE.B   LABEL_BUF,DH7      ; get seg length
            BTST     #REVERSE,SR
            JMPR.S   CC,c925_2          ; if reverse is false, don't reverse
            CLR      D4                 ; PARITY bits
            MOVE.B   DH7,DL7            ; set loop up for 4 or 6 passes
c925_1:     LSR.B    DL3                ; divide PARITY by 2, shifting remainder
            ADDC     D4,D4              ; into X, then mult. D4 by 2 and then add X
            DJNZ.B   DL7,c925_1
            MOVE     D4,D3              ; AT EXIT  DL3 ==> PARITY PATTERN
                                        ; next, test for valid segment type
c925_2:     CMP.B    #4,DH7             ; if seg length is 4 check upc d & ean8
            JMPR.S   EQ,c925_30
            BTST     #5,D3              ; this shortens the search path. If bit #5
            JMPR.S   CS,c925_13         ; is set go jump to UPCE/UPCA_R segments.
;*
            CMP.B    #0,DL3             ; seg type 00  ==> UPC_A,L ?
            JMPR.S   NE,c925_3          ; no? try next one
            MOVE.B   #$3A,DL3           ; yes? Exit with seg type in DL3
            RET                         ; and quit to main algorithm
;*
            BTST     #UPCD,DECODER1     ; are upc d label's active?
            JMPR.S   CC,c925_4
c925_3:     CMP.B    #07,DL3            ; 07  ==> UPC_D
            JMPR.S   NE,c925_4
            RET
;*
            BTST     #EAN,DECODER1
            JMPR.S   CC,c925_24
c925_4:     CMP.B    #$08,DL3           ; 08  ==> EAN13_L1
            JMPR.S   NE,c925_5
            MOVE.B   #$01,DL1           ; seg type EAN13-L? do special routine
            JMPR.S   c925_25            ; to add embedded char at seg start
c925_5:     CMP.B    #$00,DL3           ; 00  ==> EAN13_L2
            JMPR.S   NE,c925_6
            MOVE.B   #$02,DL1
            JMPR.S   c925_25
c925_6:     CMP.B    #$0E,DL3           ; 0E  ==> EAN13_L3
            JMPR.S   NE,c925_7
            MOVE.B   #$03,DL1
            JMPR.S   c925_25
c925_7:     CMP.B    #$13,DL3           ; 13  ==> EAN13_L4
            JMPR.S   NE,c925_8
            MOVE.B   #$04,DL1
            JMPR.S   c925_25
c925_8:     CMP.B    #$15,DL3           ; 15  ==> EAN13_L7
            JMPR.S   NE,c925_9
            MOVE.B   #$07,DL1
            JMPR.S   c925_25
c925_9:     CMP.B    #$16,DL3           ; 16  ==> EAN13_L8
            JMPR.S   NE,c925_10
            MOVE.B   #$08,DL1
            JMPR.S   c925_25
c925_10:    CMP.B    #$19,DL3           ; 19  ==> EAN13_L5
            JMPR.S   NE,c925_11
            MOVE.B   #$05,DL1
            JMPR.S   c925_25
c925_11:    CMP.B    #$1A,DL3           ; 1A  ==> EAN13_L9
            JMPR.S   NE,c925_12
            MOVE.B   #$09,DL1
            JMPR.S   c925_25
c925_12:    CMP.B    #$1C,DL3           ; 1C  ==> EAN13_L6
            JMPR.S   NE,c925_13
            MOVE.B   #$06,DL1
            JMPR.S   c925_25
;*
c925_13:    CMP.B    #$23,DL3           ; 23  ==> UPC_E6
            JMPR.S   NE,c925_14
```

```
          MOVE.B    #$06,DL1           ; If UPC_E segment type do special routine
          JMPR.S    c925_28            ; to add embedded char at seg end
c925_14:  CMP.B     #$25,DL3           ; 25  ==> UPC_E9
          JMPR.S    NE,c925_15
          MOVE.B    #$09,DL1
          JMPR.S    c925_28
c925_15:  CMP.B     #$26,DL3           ; 26  ==> UPC_E5
          JMPR.S    NE,c925_16
          MOVE.B    #$05,DL1
          JMPR.S    c925_28
c925_16:  CMP.B     #$29,DL3           ; 29  ==> UPC_E8
          JMPR.S    NE,c925_17
          MOVE.B    #$08,DL1
          JMPR.S    c925_28
c925_17:  CMP.B     #$2A,DL3           ; 2A  ==> UPC_E7
          JMPR.S    NE,c925_18
          MOVE.B    #$07,DL1
          JMPR.S    c925_28
c925_18:  CMP.B     #$2C,DL3           ; 2C  ==> UPC_E6
          JMPR.S    NE,c925_19
          MOVE.B    #$04,DL1
          JMPR.S    c925_28
c925_19:  CMP.B     #$01,DL3           ; 31  ==> UPC_E3
          JMPR.S    NE,c925_20
          MOVE.B    #$03,DL1
          JMPR.S    c925_28
c925_20:  CMP.B     #$02,DL3           ; 32  ==> UPC_E2
          JMPR.S    NE,c925_21
          MOVE.B    #$02,DL1
          JMPR.S    c925_28
c925_21:  CMP.B     #$04,DL3           ; 31  ==> UPC_E1
          JMPR.S    NE,c925_22
          MOVE.B    #$01,DL1
          JMPR.S    c925_28
c925_22:  CMP.B     #$08,DL3           ; 38  ==> UPC_E0
          JMPR.S    NE,c925_23
          CLR.B     DL1
          JMPR.S    c925_28
;*
c925_23:  CMP.B     #$3F,DL3           ; 3F  ==> UPC_A,R
          JMPR.S    NE,c925_24
          RET
;*
c925_24:  CLR.B     DL3                ; error, no such segment type !!!
          RET
;*                                       ADD NEW CHAR ONTO EAN13_L SEGMENTS
c925_25:  MOVE.B    #$1D,DL3           ; add segment i.d.
          BTST      #REVERSE,SR        ; if label is reversed, treat as UPC_E
          JMPR.S    CS,c925_29
c925_26:  MOVE.B    #06,DL7            ; loop counter
          MOVE      #LABEL_BUF+7,A2 ; get segment buffer END +1
          MOVE      #LABEL_BUF+8,A1 ; pointer to seg string end + 2
c925_27:  MOVE.B    -(A2),-(A1)        ; move char's over by 1 position
          DJNZ.B    DL7,c925_27
          MOVE.B    DL1,-(A1)          ; add on embedded char
          ADD.B     #01,LABEL_BUF      ; update string count
          RET
;*                                       ADD NEW CHAR ONTO UPC_E SEGMENTS
c925_28:  MOVE.B    #$3E,DL3
          BTST      #REVERSE,SR        ; if label is reversed, treat as EAN13_L
          JMPR.S    CS,c925_26
c925_29:  MOVE.B    DL1,LABEL_BUF+7 ; move embedded char to seg end
          ADD.B     #01,LABEL_BUF
          RET
;*
c925_30:  BTST      #EAN,DECODER1
          JMPR.S    CC,c925_32
          CMP.B     #00,DL3            ; 00  ==> EAN8_L
          JMPR.S    NE,c925_31
          MOVE.B    #$19,DL3
          RET
c925_31:  CMP.B     #$0F,DL3           ; 0F  ==> EAN8_R
          JMPR.S    NE,c925_32
          MOVE.B    #$18,DL3
          RET
;*
c925_32:  BTST      #UPCD,DECODER1
          JMPR.S    CC,c925_38
          CMP.B     #03,DL3            ; 03  ==> UPC_D6
          JMPR.S    NE,c925_33
```

```
          RET
c925_33:  CMP.B     #05,DL3        ; 05  ==> UPC_D5
          JMPR.S    NE,c925_34
          RET
c925_34:  CMP.B     #06,DL3        ; 06  ==> UPC_D2
          JMPR.S    NE,c925_35
          RET
c925_35:  CMP.B     #09,DL3        ; 09  ==> UPC_D3
          JMPR.S    NE,c925_36
          RET
c925_36:  CMP.B     #0A,DL3        ; 0A  ==> UPC_D4
          JMPR.S    NE,c925_37
          RET
c925_37:  CMP.B     #0C,DL3        ; 0C  ==> UPC_D1
          JMPR.S    NE,c925_38
          RET
;*
c925_38:  CLR.B     DL3            ; error, no such segment type !!!
          RET                      ; AT EXIT  DL3  ==> PARITY PATTERN
                                   ;               ==> FF if error
                                   ;



;*************************************************************************
;
;*
;* CHECK FOR SUPPLIMENTAL ADDON SEGMENT. If an addon segment has not been
;* found, check if the latest segment found is UPC_E or UPC_A,R or EAN8,R.
;* If not one of these, quit. If one of these, then try to decode an addon
;* according to the rules:
;*  SEG_TYPE        FWD_DECODE   REVERSE   IAPTR                    ADDON DIRECTION
;*  UPC_E              false       false    current_margin - 34       reverse
;*  UPC_E              true        false    current_margin + 34       foreward
;*  UPC_A,R/EAN8,R     true        true     IPTR                      foreward
;*  UPC_A,R/EAN8,R     true        false    current_margin            reverse
;*  UPC_A,R/EAN8,R     false       false    current_margin            foreward
;*
c926:     MOVE      IPTR,A5
          MOVE      A5,IA          ; save IPTR
          CMP.B     #$30,DL3       ; seg type UPC_E
          JMPR.S    NE,c926_4
          BTST      #REVERSE,SR
          JMPR.S    CS,c926_9      ; if reverse is true here, quit
          BTST      #FWD_DECODE,SR ; reverse = false !
          JMPR.S    CC,c926_2      ; fwd_decode = false ?
          MOVE      CURRENT_MARGIN,A5
          SUB       #68,A5         ; IA := current_margin - 34e
          CMP       #CDATA,A5      ; jump to c932 (look for backward addon)
          JMPR.S    GE,c926_1
          ADD       #WIDTH,A5
c926_1:   MOVE      A5,IPTR
          JMPA      c932
c926_2:   MOVE      CURRENT_MARGIN,A5
          ADD       #68,A5         ; if reverse = false and fwd_decode = true
          CMP       #CDEND,A5      ; IA := current_margin + 34e
          JMPR.S    LT,c926_3      ; jump to c929 (look for foreward addon)
          SUB       #WIDTH,A5
c926_3:   MOVE      A5,IPTR
          JMPR.S    c929
c926_4:   CMP.B     #$3F,DL3       ; test for UPC_A,R
          JMPR.S    EQ,c926_5
          CMP.B     #$0F,DL3       ; test for EAN8,R
          JMPR.S    NE,c926_9      ; no? quit
c926_5:   BTST      #FWD_DECODE,SR ; we get here if either EAN8,R or UPC_A,R
          JMPR.S    CC,c926_7      ; fwd_decode = true  ?
          BTST      #REVERSE,SR    ; reverse = true ?
                                   ; IA := IPTR
          JMPR.S    CS,c929
c926_6:   MOVE      CURRENT_MARGIN,A5
          MOVE      A5,IPTR        ; if reverse = false and fwd_decode = true
          JMPA      c932           ; jump to c932
c926_7:   BTST      #REVERSE,SR
          JMPR.S    CS,c926_9      ; if reverse = false and fwd_decode = false
          MOVE      CURRENT_MARGIN,A5
          MOVE      A5,IPTR        ; IA := current_margin
          JMPR.S    c929
c926_8:   MOVE      IA,A5          ; AT EXIT   restore IPTR
          MOVE      A5,IPTR
```

```
c926_9:   RET
;*
;* TEST FOR ADDON MARGIN FOREWARD. ( FWD_ADDON equals TRUE ) Test that element
;* (IA)/[element(IA+1) + element(IA+2) + element(IA+3)] is greater than the
;* minimum margin ratio (5.5/4) and is less than the maximum margin ratio (3).
;* If so, then test that element(IA+3)/element(IA+2) is greater than the
;* minimum guard bar addon ratio (3/2) and is less than the maximum guard bar
;* ratio (5/2). If so then check that element(IA+2)/element(IA+1) is less
;* than the maximum like element ratio (3/2) and is greater than 1/max like
;* element ratio (2/3). If okay, then set last_width to [element(IA+1) +
;* 2 * element(IA+2) + element(IA+3)] * 9/5, set IA to IA + frame_width. If
;* any test fails, exit the algorithm and return. Note that 9/5 = 1.80; we
;* have approximated this as 29/16 (=1.812).
;*
c929:     MOVE      (A5),D2         ; D2 = e0
          CALLA     TI2
          MOVE      (A5),D1         ; D1 = e1
          CALLA     TI2
          ADD       (A5),D1         ; D1 = e1 + e2
          CALLA     TI2
          ADD       (A5),D1         ; D1 = e1+e2+e3
          MOVE      D1,D0
          ADD       D1,D1
          ADD       D0,D1           ; D1 = CW*3 ; D0 = CW
          CMP       D1,D2           ; e0 > 3*CW  ?
          JMPR.S    MI,c926_8       ; quit
;*
          LSR       #1,D1           ; CW*1.5   REM D1 = 3*CW
          MOVE      D0,D7           ; CW
          ASL       #2,D7           ; CW*4
          ADD       D7,D1           ; CW*5.5
          LSR       #2,D1           ; D1 = CW*5.5/4
          CMP       D1,D2           ; e0 > CW*5.5/4
          JMPR.S    LT,c926_8       ; quit
;*
          MOVE      (A5),D1         ; D1 = e3
          CALLA     TI_2
          MOVE      (A5),D2
          MOVE      D2,D3           ; D3 = e2
          ADD       D2,D2           ; D2 = e2*2
          ADD       D3,D2           ; D2 = e2*3
          LSR       #1,D2           ; D2 = e2*3/2
          CMP       D2,D1           ; e3 > e2*3/2
          JMPR.S    LT,c926_8       ; quit
;*
          ADD       D3,D2           ; D2 = e2*3/2 + e2 = e*5/2
          CMP       D2,D1           ; e3 < e2*5/2
          JMPR.S    GT,c926_8       ; quit
;*
          ADD       D3,D0           ; D0 = e1+e2+e3 + e2
;*
          MOVE      D3,D2
          ADD       D3,D3
          ADD       D3,D2           ; D2 = e2*3
                                    ; D3 = e2*2
          CALLA     TI_2
          MOVE      (A5),D1         ; D1 = e1
          ADD       D1,D1           ; D1 = e1*2
          CMP       D1,D2           ; e2*3 > e1*2        ( e2/e1 > 2/3 )
          JMPR.S    LT,c926_8
;*
          ADD       (A5),D1         ; D1 = e1*3
          CMP       D1,D3           ; e2*2 < e1*3        ( e2/e1 < 3/2 )
          JMPR.S    GT,c926_8       ; quit
;*
          MOVE      #9,A0
          MULU      A0,D0
          MOVE      #5,A0
          DIVU      A0,D0           ; D0 = CW*9/5
          MOVE      D0,A3
          MOVE      IPTR,A5
          ADD       #8,A5           ; IA = IA + frame_width
          CMP       #GDEND,A5
          JMPR.S    LT,c929_1
          SUB       #WIDTH,A5
c929_1:   MOVE      A5,IPTR
;*
;* DECODE ADDON CHARACTERS AND FORM ADDON SEGMENT FOREWARD.
;*
c930:     CALLA     c921            ; get addon char
```

```
                                        ; AT EXIT   DH1 ==> CHARACTER
                                        ;           DL1 ==> PARITY BIT
                                        ;           DO  ==> CHAR_WIDTH
          MOVE      IPTR,A5
          CALLA     TI_4
          ADD       (A5),D0             ; D0 = CW + (e-2)
          CALLA     TI_2
          ADD       (A5),D0             ; D0 = CW + (e-2) + (e-3) = new CW
          MOVE      A3,D4               ; get last_width
          LSR       #2,D4
          ADD       A3,D4               ; LW*5/4
          CMP       D4,D0               ; CW > 5/4*LW
          JMPR.S    GT,c926_8           ; quit
;*
          MOVE      D0,D4
          LSR       #2,D4
          ADD       D0,D4               ; D4 = CW*5
          CMP       D4,A3               ; LW > CW*5/4
          JMPR.S    GT,c926_8           ; quit
;*
          MOVE      #LABEL_BUF,A4
;*
          MOVE.B    #1,(A4)+
          MOVE.B    DH1,(A4)+           ; include first addon char
          MOVE      D0,A3
          MOVE.B    DL1,PARITY
;*
c930_1:   MOVE      IPTR,A5
          ADD       #12,A5
          CMP       #GDEND,A5
          JMPR.S    LT,c930_2
          SUB       #WIDTH,A5
c930_2:   MOVE      A5,IPTR
          CALLA     c921                ; get addon char
                                        ; AT EXIT   DH1 ==> CHARACTER
                                        ;           DL1 ==> PARITY BIT
                                        ;           DO  ==> CHAR_WIDTH
          MOVE      IPTR,A5
          CALLA     TI_2
          ADD       (A5),D0             ; D0 = CW + (e-1)
          CALLA     TI_2
          ADD       (A5),D0             ; D0 = CW + (e-1) + (e-2)
          MOVE      A3,D4               ; get last_width
          LSR       #2,D4
          ADD       A3,D4               ;
          CMP       D4,D0               ; CW > 5/4*LW
          JMPA      GT,c935             ; quit
;*
          MOVE      D0,D4
          LSR       #2,D4
          ADD       D0,D4
          CMP       D4,A3               ; LW > CW*5/4
          JMPA      GT,c935             ; quit
;*
          MOVE.B    DH1,(A4)+           ; add addon char to addon seg string
          ADD.B     #1,LABEL_BUF
          MOVE      D0,A3
          ASL.B     #1,PARITY
          ADD.B     DL1,PARITY
          MOVE.B    LABEL_BUF,DL7
          CMP.B     ADDLL,DL7           ; DL7 < #ADD_LENGTH
          JMPR.S    LT,c930_1
          JMPA      c935
;*
;* TEST FOR ADDON MARGIN BACKWARD. ( FWD_ADDON equals FALSE) Check that
;* element(iA)/element(iA-1) + element(iA-2) + element(iA-3) is greater than
;* the minimum margin ratio and less than the maximum margin ratio. If ok,
;* then test that element(iA-3)/element(iA-2) is greater than minimum addon
;* guard bar ratio and less than maximum addon guard bar ratio. If this is
;* okay, check that element(iA-2)/element(iA-1) is less than the maximum like
;* element ratio and greater than 1/that ratio. If okay set last_width to
;* [element(iA-1) + 2*element(iA-2) + element(iA-3)]*9/5 and move iA to
;* iA - (2*frame_width - 1).
c932:     MOVE      (A5),D2             ; D2 = e0
          CALLA     TI_2
          MOVE      (A5),D1             ; D1 = (e-1)
          CALLA     TI_2
          ADD       (A5),D1             ; D1 = (e-1) + (e-2)
          CALLA     TI_2
          ADD       (A5),D1             ; D1 = (e-1)+(e-2)+(e-3)
```

```
            MOVE    D1,D0
            ADD     D1,D1
            ADD     D0,D1           ; D1 = CW*3
            CMP     D1,D2           ; e0 < 3*CW        ( e0/e1+e2+e3 < 3 )
            JMPA    GT,c926_8       ; quit
;*
            LSR     #1,D1
            MOVE    D0,D7
            ASL     #2,D7
            ADD     D7,D1
            LSR     #2,D1           ; D1 = CW*5.5/4
            CMP     D1,D2           ; e0 > CW*5.5.4
            JMPA    LT,c926_8       ; quit
;*
            MOVE    (A5),D1         ; D1 = (e-3)
            CALLA   T12
            MOVE    (A5),D2         ; D2 = (e-2)
            MOVE    D2,D3           ; D3 = 2*(e-2)
            ADD     D2,D2
            ADD     D3,D2           ; D2 = 3*(e-2)
            LSR     #1,D2
            CMP     D2,D1           ; (e-3) > (e-2)*3/2
            JMPA    LT,c926_8       ; quit
;*
            ADD     D3,D2           ; D2 = (e-3)*5/2
            CMP     D2,D1           ; (e-3) < (e-2)*5/2
            JMPA    GT,c926_8       ; quit
;*
            ADD     D3,D0           ; D0 = (e-1)+2*(e-2)+(e-3)
;*
            MOVE    D3,D2
            ADD     D3,D3
            ADD     D3,D2           ; D2 = (e-2)*3
                                    ; D3 = (e-2)*2
            CALLA   T12
            MOVE    (A5),D1         ; D1 = (e-1)
            ADD     D1,D1           ; D1 = (e-1)*2
            CMP     D1,D2           ; (e-2)*3 > (e-1)*2    ( e2/e1 > 2/3 )
            JMPA    LT,c926_8
;*
            ADD     (A5),D1         ; D1 = (e-1)*3
            CMP     D1,D3           ; (e-2)*2 < (e-1)*3    ( e2/e1 < 3/2 )
            JMPA    GT,c926_8       ; quit
;*
            MOVE    #9,A0
            MULU    A0,D0
            MOVE    #5,A0
            DIVU    A0,D0
            MOVE    D0,A3
            MOVE    IPTR,A5
            SUB     #14,A5          ; iA = iA - (2*frame_width - 1 )
            CMP     #GDATA,A5
            JMPR.S  GE,c932_1
            ADD     #WIDTH,A5
c932_1:     MOVE    A5,IPTR
;*
;* DECODE ADDON CHARACTERS AND FORM ADDON SEGMENT BACKWARD.
;*
c933:       CALLA   c921            ; get addon char
                                    ; AT EXIT   DH1 ==> CHARACTER
                                    ;           DL1 ==> PARITY BIT
                                    ;           D0  ==> CHAR_WIDTH
            MOVE    IPTR,A5
            ADD     #10,A5
            CMP     #GDEND,A5
            JMPR.S  LT,c933_1
            SUB     #WIDTH,A5
c933_1:     ADD     (A5),D0         ; D0 = CW + (e5)
            CALLA   T12
            ADD     (A5),D0         ; D0 = CW + (e5) + (e6)
            MOVE    A3,D4
            LSR     #2,D4
            ADD     A3,D4
            CMP     D4,D0           ; CW > 5/4*LW
            JMPA    GT,c926_8       ; quit
;*
            MOVE    D0,D4
            LSR     #2,D4
            ADD     D0,D4
            CMP     D4,A3           ; LW > CW*5/4
```

```
        JMPA     GT,c926_8          ; quit
;*
        MOVE     #LABEL_BUF,A4
;*
        MOVE.B   #1,(A4)+
        MOVE.B   DH1,(A4)+          ; include first addon char
        MOVE     D0,A3
        MOVE.B   DL1,PARITY
;*
c933_2: MOVE     IPTR,A5
        SUB      #12,A5
        CMP      #GDATA,A5
        JMPR.S   GE,c933_3
        ADD      #WIDTH,A5
c933_3: MOVE     A5,IPTR
        CALLA    c921               ; get addon char
                                    ; AT EXIT   DH1 ==> CHARACTER
                                    ;           DL1 ==> PARITY BIT
                                    ;           D0  ==> CHAR_WIDTH
        MOVE     IPTR,A5
        ADD      #08,A5
        CMP      #GDEND,A5
        JMPR.S   LT,c933_4
        SUB      #WIDTH,A5
c933_4: ADD      (A5),D0            ; D0 = CW + (e4)
        CALLA    T12
        ADD      (A5),D0            ; D0 = CW + (e4) + (e5)
        MOVE     A3,D4
        LSR      #2,D4
        ADD      A3,D4
        CMP      D4,D0              ; CW > 5/4*LW
        JMPR.S   GT,c935            ; quit
;*
        MOVE     D0,D4
        LSR      #2,D4
        ADD      D0,D4
        CMP      D4,A3              ; LW > CW*5/4
        JMPR.S   GT,c935            ; quit
;*
        MOVE.B   DH1,(A4)+          ; add addon char to addon seg string
        ADD.B    #1,LABEL_BUF
        MOVE     D0,A3
        ASL.B    #1,PARITY          ; update PARITY pattern
        ADD.B    DL1,PARITY
        MOVE.B   LABEL_BUF,DL7
        CMP.B    ADDLL,DL7          ; test if addon string is too long
        JMPR.S   LT,c933_2          ; DL7 < #ADDON_LENGTH ?
;*
;* CALCULATE SEGMENT CHECK SUM. (REF 9.3.35 - 9.3.37). If length of addon
;* segment is 2, calculate the check sum as the mod 4 of the two digit addon.
;* If length of the addon segment is 5, calculate the check sum as [(3 * sum
;* of digits 1, 3 and 5 of addon string) + (9 * sum of digits 2 and 4)] mod 10.
;* Index the PARITY bits into the table. If the index PARITY pattern equals
;* the calculated check sum, addon is true. Store the addon string in Add_1
;* and return, else just return.
;*
c935:   MOVE.B   PARITY,DL3         ; get PARITY pattern
        MOVE     #LABEL_BUF,A0      ; load base addresses for label buffer
        CLR      D1                 ; prep. working reg.
        MOVE.B   (A0)+,DL7          ; fetch addon seg length
        CMP.B    #2,DL7             ; if addon seg length = 2
        JMPR.S   NE,c935_2
        MOVE.B   (A0),DL1           ; get N1
        ADD      D1,D1              ; D1 = 2*N1
        MOVE     D1,D0
        ASL      #2,D0              ; D0 = 4*(2*N1) = 8*N1
        ADD      D0,D1              ; D1 = 10*N1
        ADD.B    1(A0),DL1          ; get N2
                                    ; D0 = ((10*N1) + N2)
        AND      #03,D0             ;      ((10*N1) + N2) MOD 4
        CMP.B    DL0,DL3
        JMPA     NE,c926_8          ; not equal? Parity error. Quit!
        MOVE.B   #$22,DH7
        CALLA    outch              ; show seg is 2 char addon
c935_1: MOVE.B   (A0)+,DH7          ; else, move 2 char addon string
        CALLA    outch
        DJNZ.B   DL7,c935_1
        JMPA     c926_8             ; end leave
c935_2: CMP.B    #5,DL7             ; seg length equal 5 ?
        JMPA     NE,c926_8          ; if not = 5, error, quit.
```

```
          MOVE.B    (A0),DL0          ; get N1
          ADD.B     2(A0),DL0         ; get N3
          ADD.B     4(A0),DL0         ; get N5
          MOVE      D0,D1
          ADD       D1,D1
          ADD       D1,D0             ; D0 := 3*(N1+N3+N5)
          MOVE.B    1(A0),DL1         ; get N2
          ADD.B     3(A0),DL1         ; get N4
          MOVE      D1,D2
          ASL       #3,D1             ; D1 = 8*(N2+N4)
          ADD       D2,D1             ; D1 := 9*(N2+N4)
          ADD       D0,D1             ; D1 = 3*(N1+N3+N5) + 9*(N2+N4)
          MOVE      #10,A4
          CLR       D0
          DIVU      A4,D0             ; calc mod 10 result
          MOVE      #add5tbl,A1
          ADD       D1,A1
          MOVE.B    (A1),DL0
          CMP.B     DL0,DL3           ; PARITY = remainder ?
          JMPA      NE,c926_8         ; not equal? Parity error, quit!
          MOVE.B    #$25,DH7          ; else,
          CALLA     outch             ; show 5 char addon seg type
c935_3:   MOVE.B    (A0)+,DH7         ; move 5 char addon string
          CALLA     outch
          DJNZ.B    DL7,c935_3
          JMPA      c926_8            ; and return to main algorithm

;///////////////////////////////////////////////////////////////////////
;/                                                                     /
;/        START THE MAIN DECODING ALGORITHM                            /
;/                                                                     /
;///////////////////////////////////////////////////////////////////////

;*
;* START OF UPC/EAN DECODER
;*
codeUPC:  BTST      #UPC,DECODER1     ; switch closed...do UPC/EAN decoders?
          JMPA      CC,no_decode      ; no, make like a tree and leave
          MOVE      LASTI,D0
          MOVE      D0,IPTR           ; yes, get back original IPTR
;*
;* LOOK FOR A MARGIN BACKWARD. If the difference between IPTR and the last
;* decode point is equal to or greater than a minimum segment size (4 char-
;* acter segment = 26 elements) then do step 9.3.10 to check for a margin in
;* the reverse direction. If a margin has been found then:
;*    - set last char width to margin scalar * ([e-1] + 2*[e-2] + [e-3])
;*    - set current margin to IPTR
;*    - set IPTR to IPTR - (2*frame_width - 1); i.e. point to the first
;*      element of the next character to be decoded.
;*    - set fwd_decode false
;*    - reset PARITY bits and segment strings to empty
;*    - set continue_decode true
;*
                                      ; get lastest wide space (in D0 ==> IPTR)
c902:     SUB       ILDP,D0           ; subtract last decode point
          JMPR.S    PL,c902_1
          NEG       D0                ; get the absolute value of calculation
c902_1:   CMP       #52,D0            ; D0 < 52 ?
          JMPA      LT,c903           ; if so, test for margin foreward
                                      ; ref 9.3.10
          MOVE      IPTR,A5
          MOVE      (A5),D0           ; get e0
          CALLA     Ti_2
          MOVE      (A5),D1           ; get [e-1]
          CALLA     Ti_2
          ADD       (A5),D1           ; get [e-2]
          ADD       (A5),D1           ; D1 = [e-1] + 2*[e-2]
          CALLA     Ti_2
          ADD       (A5),D1           ; D1 = [e-1] + 2*[e-2] + [e-3]
          MOVE      D1,D2
          MOVE      D1,D5
          LSR       #1,D2             ; CW/2
          ADD       D1,D2             ; CW/2 + CW
          ASL       #2,D5             ; REM! D5 = CW*4
          ADD       D5,D2             ; CW/2 + CW + CW*4  =  CW*5.5
          LSR       #2,D2             ; 5.5*CW/4
          CMP       D2,D0             ; D0 < D2 ?
          JMPA      LT,c903           ; REM! D1 = [e-1] + 2*[e-2] + [e-3]
;*                                           D5 = 5*CW
```

```
        MOVE    (A5),D0         ; get [e-3] = D0
        CALLA   T16
        MOVE    (A5),D2         ; get [e-1] = D2
        MOVE    D0,D3
        ADD     D3,D3
        ADD     D0,D3           ; [e-3]*3
        LSR     #1,D3           ; [e-3]*3/2
        CMP     D3,D2           ; [e-1] > [e-3]*3/2 ?
        JMPA    GT,c903
;*
        MOVE    D2,D3           ; REM! D0 = [e-3] / D2 = [e-1]
        ADD     D3,D3
        ADD     D2,D3           ; [e-1]*3
        LSR     #1,D3           ; [e-1]*3/2
        CMP     D3,D0           ; [e-3] > [e-1]*3/2 ?
        JMPA    GT,c903
;*                               REM D5 = CW*4
        ADD     D1,D5           ; D5 = CW*4 + CW = CW*5
        ADD     D1,D1           ; CW*2
        ADD     D5,D1           ; D1 = 2*CW + 5*CW = 7*CW
        LSR     #2,D1           ; D1 = 7*CW/4
        MOVE    D1,A3           ; store last_width
        BCLR    #FWD_DECODE,SR
        MOVE    IPTR,A5
        MOVE    A5,CURRENT_MARGIN
        SUB     #14,A5
        CMP     #GDATA,A5
        JMPR.S  GE,c902_2
        ADD     #WIDTH,A5
c902_2: MOVE    A5,IPTR         ; AT EXIT   A5 ==> IPTR
;*
;* GET BACKWARD SEGMENT ALGORITHM. Test if data is available in the backward
;* direction (IPTR >= last_decode_point +1). If not available, quit the de-
;* coding algorithm and do step 9.3.14. In the main algorithm use steps 9.3.21
;* through 9.3.24 to get a segment character. If character is valid:
;*     - add character to the segment string
;*     - shift the PARITY map left one bit and add 1 if even PARITY
;*     - set last_char_width to current_char_width
;*     - set IPTR to IPTR - frame_width
;* If character is not valid do step 9.3.14. At the end of each "loop" through
;* main algorithm test the segment buffer to see if the segment string is
;* greater than 6 characters in length, if it is set last_decode_point to
;* current margin and enter 9.3.3.
;*
c911a:  CLR     LABEL_BUF
        MOVE    #LABEL_BUF+1,A4
        CLR.B   PARITY
c911:   CALLA   c921            ; get a character
c912:   CALLA   c924            ; check character
                                ; on return from c924 D0  ==> CHAR_WIDTH
                                ;                     DH1 ==> CHARACTER
                                ;                     DL1 ==> PARITY BIT
                                ;                     D6  ==> 0000 if ok
                                ;                             00FF if small
                                ;                             FF00 if large
        CMP     #0,D6           ; char too large or too small
        JMPR.S  NE,c914         ; if so, quit to 9.3.14
        MOVE    D0,A3           ; last_width := char_width
        MOVE.B  DH1,(A4)+
        ADD.B   #1,LABEL_BUF
        ASL.B   #1,PARITY       ; shift PARITY bit left by 1
        ADD.B   DL1,PARITY      ; add new PARITY bit
        MOVE    IPTR,A5
        SUB     #08,A5
        CMP     #GDATA,A5
        JMPR.S  GE,c912_1
        ADD     #WIDTH,A5
c912_1: MOVE    A5,IPTR
c913:   BTST    #11,LABEL_BUF   ; label_buf < 7 ?
        JMPR.S  CC,c911         ; if fail ==> ref. 9.3.15
        JMPR.S  c915            ; re-enter the algorithm foreward
;*
;* TEST SEGMENT STRING, MAKING AND STORING IT AS A VALID SEGMENT TYPE IF
;* POSSIBLE. REF 9.3.14
;*
c914:   MOVE.B  LABEL_BUF,DL7
        CMP.B   #06,DL7         ; if segment length is not 4 or 6
        JMPR.S  EQ,c914_1       ; goto c915
        CMP.B   #04,DL7
        JMPR.S  NE,c915
```

```
c914_1:  BTST     #1,IPTR          ; if IPTR isn't pointing at a bar
         JMPR.S   CS,c915          ; goto c915 (framing error)
         CMP      #$00FF,D6        ; if not too small, quit
         JMPR.S   NE,c915          ; (isn't a center band)
         CMP.B    #$01,DH1         ; if char isn't ambiguous (1, 2, 7, or 8)
         JMPR.S   EQ,c914_2        ; goto c915 (isn't a center band)
         CMP.B    #$02,DH1
         JMPR.S   EQ,c914_2
         CMP.B    #$07,DH1
         JMPR.S   EQ,c914_2
         CMP.B    #$08,DH1
         JMPR.S   NE,c915
c914_2:  MOVE     D0,A3
         MOVE     IPTR,A5
         SUB      #02,A5           ; decrement IPTR by 1
         CMP      #CDATA,A5
         JMPR.S   GE,c914_3
         ADD      #WIDTH,A5
c914_3:  MOVE     A5,IPTR
         CALLA    c921             ; then, get a new char
         CALLA    c924             ; and test it
         CMP      #0,D6            ; if it fails, quit to 9.3.15
         JMPR.S   NE,c915
         CMP.B    #$01,DH1         ; then, test if char is an ambiguous one
         JMPR.S   EQ,c914_4
         CMP.B    #$02,DH1
         JMPR.S   EQ,c914_4
         CMP.B    #$07,DH1
         JMPR.S   EQ,c914_4
         CMP.B    #$08,DH1
         JMPR.S   NE,c915          ; if not, quit to 9.3.15
c914_4:  BCLR     #REVERSE,SR      ; if okay to here
         CALLA    c925             ; get parity pattern with reverse false
                                   ; RE-ENTER WITH  DL3 ==> PARITY PATTERN
                                   ;                     FF if error
         CMP.B    #$00,DL3         ; test if valid segment type
         JMPR.S   EQ,c915          ; not a valid seg? Quit!
         MOVE.B   DL3,DH7          ; send seg type
         CALLA    outch
         MOVE     #LABEL_BUF,A2    ; get seg length
         MOVE.B   (A2)+,DL7        ; REM: A2 points to first char.
                                   ; then, send the segment
         CMP.B    #$3F,DL3         ; test seg type; is it UPC_A,R ?
         JMPR.S   EQ,c914_5
         CMP.B    #$0C,DL3         ; or UPC_D1 ?
         JMPR.S   EQ,c914_5
         CMP.B    #$18,DL3         ; or EAN8,R ?
         JMPR.S   NE,c914_7        ; if not one of these, skip next step
c914_5:  CLR.B    DH7              ; if UPC_A,R or UPC_D1 or EAN8,R move
         ADD      D7,A2            ; move pointer to string end + 1
c914_6:  MOVE.B   -(A2),DH7
         CALLA    outch
         DJNZ.B   DL7,c914_6
         JMPR.S   c914_8
c914_7:  MOVE.B   (A2)+,DH7        ; else, move string foreward
         CALLA    outch
         DJNZ.B   DL7,c914_7
;*
c914_8:  MOVE.B   ADDLL,DL0        ; look for addon seg ?
         CMP.B    #0,DL0
         CALLA    NZ,c926
;*
c915:    MOVE     CURRENT_MARGIN,A5
         MOVE     A5,ILDP
         MOVE     A5,IPTR
;*
;* LOOK FOR A MARGIN FOREWARD. If we failed test c902, we end up here. If
;* there are less than one frame_width elements available to be examined,
;* wait (bar room). If we fail here, quit (didn't find a margin in either
;* direction). If enough then do step 9.3.10 to test for a margin in the
;* foreward direction. If a margin is found, then.
;*     - set last char width to margin scaler * (e1 + 2*e2 + e3)
;*     - set current margin to IPTR
;*     - set IPTR to IPTR + frame_width ; i.e. point to the first
;*       element of the next character to be decoded.
;*     - set fud_decode true
;*     - reset parity bits and segment strings to empty
;*
c903:    MOVE     IPTR,A5          ; ref 9.3.10
         MOVE     (A5),D0          ; get e0
```

```
        CALLA   112
        MOVE    (A5),D1         ; get e1
        CALLA   112
        ADD     (A5),D1         ; get e2
        ADD     (A5),D1         ; D1 = e1 + 2*e2
        CALLA   T12
        ADD     (A5),D1         ; D1 = e1 + 2*e2 + e3
        MOVE    D1,D2
        MOVE    D1,D5
        LSR     #1,D2           ; CV/2
        ADD     D1,D2           ; CV/2 + CV
        ASL     #2,D5           ; D5 = CV*4
        ADD     D5,D2           ; CV/2 + CV + CV*4  =  CV*5.5
        LSR     #2,D2           ; 5.5*CV/4
        CMP     D2,D0           ; D0 < D2 ?
        JMPA    LT,no_decode    ; rem  D1 = e1 + 2*e2 + e3
                                ;      D5 = CV*4
;*
        MOVE    (A5),D0         ; get e3 = D0
        CALLA   T1_4
        MOVE    (A5),D2         ; get e1 = D2
        MOVE    D0,D3
        ADD     D3,D3
        ADD     D0,D3           ; e3*3
        LSR     #1,D3           ; e3*3/2
        CMP     D3,D2           ; e1 > e3*3/2 ?
        JMPA    GT,no_decode
;*
        MOVE    D2,D3           ; rem  D0 = e3 / D2 = D1
        ADD     D3,D3
        ADD     D2,D3           ; e1*3
        LSR     #1,D3           ; e1*3/2
        CMP     D3,D0           ; e3 > e1*3/2 ?
        JMPA    GT,no_decode
                                ; rem  D5 = CV*4
;*
        ADD     D1,D5           ; D5 = CV*4 + CV
        ADD     D1,D1           ; CV*2
        ADD     D5,D1           ; D1 = 2*CV + 5*CV = 7*CV
        LSR     #2,D1           ; D1 = W*7/4
        MOVE    D1,A3
        BSET    #FWD_DECODE,SR
        MOVE    IPTR,A5
        MOVE    A5,CURRENT_MARGIN
        MOVE    A5,ILDP
        ADD     #08,A5
        CMP     #CDEND,A5
        JMPR.S  LT,c903_1
        SUB     #WIDTH,A5
c903_1: MOVE    A5,IPTR         ; AT EXIT    A5 ==> IPTR
;*
;* GET FOREWARD SEGMENT ALGORITHM. Do steps 9.3.21 thru 9.3.24 to get a
;* possible character. If successful:
;*     - add character to the segment string
;*     - shift the parity pattern left 1 bit and add new parity bit
;*     - set last_width to current character width
;*     - add frame_width to IPTR
;*
c916a:  CLR     LABEL_BUF
        MOVE    #LABEL_BUF+1,A4
        CLR.B   PARITY
c916:   CALLA   room
c917:   CALLA   c921            ; get a character
        CALLA   c924            ; check character
                                ; on return  D0   ==> CHAR_WIDTH
                                ;            DH1  ==> CHARACTER
                                ;            DL1  ==> PARITY BIT
                                ;            D6   ==> 0000 if ok
                                ;                     00FF if small
                                ;                     FF00 if large
                                ;
        CMP     #0,D6           ; char too large or too small
        JMPR.S  NE,c919         ; if so, quit to 9.3.19
        MOVE    D0,A3           ; last_width := char_width
        MOVE.B  DH1,(A4)+       ; insert char
        ADD.B   #1,LABEL_BUF
        ASL.B   #1,PARITY       ; shift PARITY bit left by 1
        ADD.B   DL1,PARITY      ; add new PARITY bit
        MOVE    IPTR,A5
        ADD     #08,A5          ; move pointer to next char
        CMP     #CDEND,A5
        JMPR.S  LT,c917_1
        SUB     #WIDTH,A5
```

```
c917_1:  MOVE     A5,IPTR
c918:    BTST     #11,LABEL_BUF
         JMPR.S   CC,c916          ; if too many char's ==> QUIT
         JMPA     no_decode
;*
;* TEST SEGMENT STRING, MAKING AND STORING IT AS A VALID SEGMENT TYPE IF
;* POSSIBLE. REF 9.3.19
;*
c919:    MOVE.B   LABEL_BUF,DL7
         CMP.B    #06,DL7          ; if segment length is not 4 or 6 QUIT
         JMPR.S   EQ,c919_1
         CMP.B    #04,DL7
         JMPA     NE,no_decode
c919_1:  BTST     #1,IPTR          ; if IPTR is pointing at a bar
         JMPR.S   CC,c920          ; goto c920, else
;*                                   SEGMENT ENDED ON A CENTER BAND
         CMP      #$00FF,D6        ; if char is not too small, quit
         JMPA     NE,no_decode
         CMP.B    #$01,DH1         ; if char isn't ambiguous (1, 2, 7, or 8)
         JMPR.S   EQ,c919_2        ; QUIT (isn't a center band)
         CMP.B    #$02,DH1
         JMPR.S   EQ,c919_2
         CMP.B    #$07,DH1
         JMPR.S   EQ,c919_2
         CMP.B    #$08,DH1
         JMPA     NE,no_decode
c919_2:  MOVE     A3,LAST_WIDTH    ; store away width of last char
         MOVE     D0,A3            ; store width of margin
         MOVE     IPTR,A5
         ADD      #02,A5           ; increment IPTR by 1
         CMP      #CDEND,A5
         JMPR.S   LT,c919_3
         SUB      #WIDTH,A5
c919_3:  MOVE     A5,IPTR
         CALLA    c921             ; get another char
         CALLA    c924             ; and test it
         CMP      #0,D6            ; if test fails, quit
         JMPA     NE,no_decode
         CMP.B    #$01,DH1
         JMPR.S   EQ,c919_4        ; test if char is ambiguous
         CMP.B    #$02,DH1
         JMPR.S   EQ,c919_4
         CMP.B    #$07,DH1
         JMPR.S   EQ,c919_4
         CMP.B    #$08,DH1
         JMPA     NE,no_decode
c919_4:  BCLR     #REVERSE,SR
         CALLA    c925             ; get parity pattern with reverse false
                                   ; RE-ENTER WITH  DL3 ==> PARITY PATTERN
                                   ;                        FF if error
         CMP.B    #$00,DL3         ; test if valid segment type
         JMPA     EQ,no_decode     ; no? Quit!
         MOVE.B   DL3,DH7          ; else, send seg type
         CALLA    outch
         MOVE     #LABEL_BUF,A2
         MOVE.B   (A2)+,DL7
                                   ; then, send segment
         CMP.B    #$3F,DL3         ; test seg type; is it UPC_A,R ?
         JMPR.S   EQ,c919_5
         CMP.B    #$0C,DL3         ; or UPC_D1 ?
         JMPR.S   EQ,c919_5
         CMP.B    #$18,DL3         ; or EAN8,R ?
         JMPR.S   NE,c919_7        ; if not one of these, skip next step
c919_5:  CLR.B    DH7              ; if UPC_A,R or UPC_D1 or EAN8,R reverse
         ADD      D7,A2            ; string (scanned it backwards).
c919_6:  MOVE.B   -(A2),DH7
         CALLA    outch
         DJNZ.B   DL7,c919_6
         JMPR.S   c919_8
c919_7:  MOVE.B   (A2)+,DH7        ; else, store seg string non-reversed
         CALLA    outch
         DJNZ.B   DL7,c919_7
;*
c919_8:  MOVE.B   ADDLL,DL0        ; look for addon seg ?
         CMP.B    #0,DL0
         CALLA    NZ,c926
;*
c919_10: MOVE     LAST_WIDTH,A3    ; get back previous last_width
         MOVE     IPTR,A5          ; else, continue decoding
         ADD      #8,A5
```

```
            CMP      #GDEND,A5
            JMPR.S   LT,c919_12
            SUB      #WIDTH,A5
c919_12:    MOVE     A5,IPTR
            MOVE     A5,ILOP
            JMPA     c916e
;*                                 SEGMENT ENDED ON A MARGIN
c920:       CMP      #$FF00,D6     ; if char is not too big, quit
            JMPR.S   NE,no_decode  ; (not a margin character)
            MOVE     IPTR,A5
            ADD      #06,A5        ; increment IPTR by 3
            CMP      #GDEND,A5
            JMPR.S   LT,c920_1
            SUB      #WIDTH,A5
c920_1:     MOVE     A5,IPTR       ; check margin, ref. 9.3.10
            MOVE     (A5),D3       ; D3 = e0
            CALLA    T1_2
            MOVE     (A5),D4       ; get [e-1]
            CALLA    T1_2
            ADD      (A5),D4
            ADD      (A5),D4
            CALLA    T1_2
            ADD      (A5),D4       ; D4 = [e-1]+2*[e-2]+[e-3]
            MOVE     D4,D5
            ASL      #2,D5
            ADD      D4,D5         ; D5 = 5*CW
            LSR      #1,D4         ; D4 = CW/2
            ADD      D5,D4         ; D4 = 5.5*CW
            LSR      #2,D4         ; D4 = CW*5.5/4
            CMP      D3,D4         ; CW*5.5/4 > e0 ?
            JMPR.S   GT,no_decode
;*
            MOVE     (A5),D3       ; D3 = [e-3]
            CALLA    T14
            MOVE     (A5),D4       ; D4 = [e-1]
            ADD      D4,D4
            ADD      (A5),D4       ; D4 = [e-1]*3
            LSR      #1,D4
            CMP      D4,D3         ; [e-3] > [e-1]*3/2
            JMPR.S   GT,no_decode
;*
            MOVE     (A5),D4       ; D4 = [e-1]
            MOVE     D3,D5
            ADD      D3,D3
            ADD      D5,D3
            LSR      #1,D3         ; D3 = [e-3]*3/2
            CMP      D3,D4         ; [e-1] > [e-3]*3/2 ?
            JMPR.S   GT,no_decode
;*
            BSET     #REVERSE,SR
            CALLA    c925          ; get parity pattern with reverse false
                                   ; RE-ENTER WITH  DL3 ==> PARITY PATTERN
                                   ;                        FF if error
            CMP.B    #$00,DL3      ; test if valid segment type
            JMPR.S   EQ,no_decode  ; no? Quit!
            MOVE.B   DL3,DH7       ; else, send seg type
            CALLA    outch
            MOVE     #LABEL_BUF,A2
            MOVE.B   (A2)+,DL7
            CMP.B    #$3F,DL3      ; else, test seg type; is it UPC_A,R ?
            JMPR.S   EQ,c920_4
            CMP.B    #$0C,DL3      ; or UPC_D1 ?
            JMPR.S   EQ,c920_4
            CMP.B    #$18,DL3      ; or EAN8,R ?
            JMPR.S   EQ,c920_4
                                   ; if not one of those, reverse the string
c920_2:     CLR.B    DH7           ; (scanned it backwards)
            ADD      D7,A2
c920_3:     MOVE.B   -(A2),DH7
            CALLA    outch
            DJNZ.B   DL7,c920_3
            JMPR.S   c920_5
c920_4:     MOVE.B   (A2)+,DH7     ; else, store seg string non-reversed
            CALLA    outch
            DJNZ.B   DL7,c920_4
;*
c920_5:     MOVE.B   ADDLL,DL0     ; look for addon seg ?
            CMP.B    #0,DL0
            CALLA    NZ,c926
;*
```

```
c920_6:  MOVE     @PTR,A5
         MOVE     A5,CURRENT_MARGIN
         MOVE     A5,@LDP
         JMPA     c903            ; start again looking for a fwd margin
```

```
          ;              0  1  2  3  4  5  6  7  8  9
xctartbl: DC.B      00,00,00,01,00,00,02,00,00,03      ; 000
          DC.B      00,04,05,00,06,00,00,00,07,00      ; 010
          DC.B      00,08,00,00,09,00,10,00,00,00      ; 020
          DC.B      00,00,00,11,00,00,12,00,00,00      ; 030
          DC.B      00,13,00,00,14,00,00,00,15,00      ; 040
          DC.B      00,00,00,00,00,00,16,00,00,00      ; 050
          DC.B      00,00,00,00,00,00,17,00,00,18      ; 060
          DC.B      00,00,19,00,20,00,00,00,00,00      ; 070
          DC.B      00,21,00,00,22,00,00,00,00,00      ; 080
          DC.B      00,00,00,00,00,00,23,00,00,00      ; 090
          DC.B      00,00,00,00,24,00,00,00,00,00      ; 100
          DC.B      00,00,00,00,00,00,00,00,00,00      ; 110
          DC.B      00,00,00,00,00,00,00,00            ; 120

xfcbar:   DC.B      '012C-D4+$A678x8x5:9x/.3x'
xrcbar:   DC.B      '389x8x7.-x54xA1D6/28z+0C'

;*
;* Find the widest bar and widest space amoung the character element(i+1)
;* though element(i+7), then multiply the wisest bar and space by the
;* threshold ratio (0.700). Set a binary number, which will represent the
;* character offset, to zero. For each element, e1 through e7, multiply
;* the pattern by 2, then incrment the pattern by 1 if the element is
;* larger than the calculated threshold (bar or space,as appropriate).
;* If no narrow space was found (space pattern = 3), logically AND the
;* calculated pattern with %1010101 ($55) to correct it for the case of
;* there being no wide spaces.
;*
c611:     MOVE      IPTR,A5
          MOVE.B    #5,DL7
          CALLA     TI2
          MOVE      (A5),D3          ; e1
          CALLA     TI2
          MOVE      (A5),D4          ; e2
c611_1    CALLA     TI2
          CMP       (A5),D3          ; e1 > e3
          JMPR.S    GE,c611_2
          MOVE      (A5),D3
c611_2:   SUB.B     #1,DL7
          JMPR.S    EQ,c611_4
          CALLA     TI2
          CMP       (A5),D4          ; e2 > e4
          JMPR.S    GE,c611_3
          MOVE      (A5),D4
c611_3:   DJNZ.B    DL7,c611_1
c611_4:   MOVE      D3,A1
          ASL       #3,A1
          ADD       D3,A1
          ADD       D3,A1
          ADD       D3,A1            ; REM   D3 ==> widest bar
          LSR       #4,A1            ; bar_bkpt = max_bar*11/16
          MOVE      D4,A2
          ASL       #3,A2
          ADD       D4,A2
          ADD       D4,A2
          ADD       D4,A2            ; REM   D4 ==> widest space
          LSR       #4,A2            ; sp_bkpt = max_space*11/16
c612:     CLR       D6
          MOVE.B    #7,DL7
          MOVE      IPTR,A5
c612_1:   ADD.B     DL6,DL6
          CALLA     TI2
          CMP       (A5),A1          ; bar_bkpt > (A5)
          JMPR      GT,c612_2
          ADD.B     #1,DL6
c612_2:   SUB.B     #1,DL7
          JMPR.S    EQ,c613
          ADD.B     DL6,DL6
          CALLA     TI2
          CMP       (A5),A2          ; sp_bkpt > (A5)
          JMPR      GT,c612_3
          ADD.B     #1,DL6
          ADD.B     #1,DH6
c612_3:   DJNZ.B    DL7,c612_1
c613:     CMP.B     #3,DH6
          AND.B     #$55,DL1
```

....

```
;*
;* Use the calculated pattern to look up the character index value, if the
;* index value is zero, the test failed (no valid character found). Quit the
;* decoder.
c614:   MOVE     #xcbartbl,A3
        CLR      D7              ; AT EXIT  DH6 ==> 3 if no wide space
        MOVE.B   DL6,DL7         ;          DL7 ==> char index
        ADD      D7,A3           ;          D3  ==> widest bar
        MOVE.B   (A3),DL7        ;          D4  ==> widest space
        RTS

;*
;* Using the calculated chacter index value, get a character from either the
;* foreward or reverse character table. If the character is not valid ("x")
;* quit the decoder.
;*
c615:   BTST     #FOREWARD,SR
        JMPR.S   CC,c615_1
        MOVE     #xfcbar,A3
        JMPR.S   c615_2
c615_1: MOVE     #xrcbar,A3
c615_2: ADD      D7,A3           ; AT EXIT  DH6 ==> 3 if no wide space
        MOVE.B   (A3),DH7        ;          DH7 ==> char or "x" if error
        CMP      #$78,DH7        ;          D3  ==> widest bar
        JMPA     EQ,c622f        ;          D4  ==> widest space
;*
;* Check the character widths. Find the narrowest bar and space and calculate
;* the sum of element(i+1) through element(i+7). If the ratio of the widest
;* bar to the narrowest bar is greater than the max element ratio (5.0)or is
;* less than the min element ratio (1.5), quit. If the ratio of the widest
;* space to the narrowest space is greater than the max element ratio, quit.
;* If wide spaces were found (DH6 NOT EQUAL 3) and the ratio of the widest
;* space to the narrowest space is less than the min element ratio, quit. If
;* the rsatio of the narrowest bar to the narrowest space or the inverse of
;* this ratio is greater then the max element narrow ratio (3.0), quit.
;*
c616:   MOVE     iPTR,A5         ; REM    D3 ==> wb    D2 ==> ws
        MOVE.B   #5,DL7
        CALLA    Ti2
        MOVE     (A5),D1         ; e1
        MOVE     D1,D0
        CALLA    Ti2
        MOVE     (A5),D2         ; e2
        ADD      D2,D0
c616_1  CALLA    Ti2
        CMP      (A5),D1         ; e1 > e3
        JMPR.S   LE,c616_2
        MOVE     (A5),D1
c616_2: ADD      (A5),D0         ; D0 = e1+e2+e3+e4+e5+e6+e7
        SUB.B    #1,DL7
        JMPR.S   EQ,c616_4
        CALLA    Ti2
        CMP      (A5),D2         ; e2 > e4
        JMPR.S   GE,c616_3
        MOVE     (A5),D2         ; AT EXIT  D0  ==> CHAR_WIDTH
c616_3: ADD      (A5),D0         ;          D1  ==> narrowest bar
        DJNZ.B   DL7,c616_1      ;          D2  ==> narrowest space
                                 ;          D3  ==> widest bar
                                 ;          D4  ==> widest space
                                 ;          DH6 ==> 3 if no wide space
                                 ;          DH7 ==> character
c617:   MOVE     D1,D5
        ASL      #2,D5
        ADD      D1,D5
        CMP      D5,D3           ; wb/nb > 5.0
        JMPA     GT,c622f
        MOVE     D1,D5
        LSR      D5
        ADD      D1,D5
        CMP      D5,D3           ; wb/nb < 1.5
        JMPA     LT,c622f
c618:   MOVE     D2,D5
        ASL      #2,D5
        ADD      D2,D5
        CMP      D5,D4           ; ws/ns > 5.0
        JMPA     GT,c622f
c619:   CMP.B    #3,DH6
        JMPR.S   EQ,c620
        MOVE     D2,D5
        LSR      D5
```

```
            ADD     D2,D5
            CMP     D5,D4           ; ws/ns < 1.5
            JMPA    LT,c622f
c620:       MOVE    D2,D5
            ADD     D2,D5
            ADD     D2,D5
            CMP     D5,D1           ; nb/ns > 3.0
            JMPA    LT,c622f
c621:       MOVE    D1,D5
            ADD     D1,D5
            ADD     D1,D5
            CMP     D5,D2           ; ns/nb > 3.0
            JMPA    LT,c622f
            MOVE.B  DH7,(A4)+       ; If okay to here, store character
            RET                     ; AT EXIT  D0  ==> char width
c622f:      CLR.B   DH7             ;          DH7 ==> char or "0" if error
            RET


codeCBAR:   BTST    #CBAR,DECODER1
            JMPA    CC,nextcode
            MOVE    lastl,A5
            MOVE    A5,IPTR
;*
;* If the sum of elements e1+e2+e3 > e0, quit (margin too small).
;* NOTE: THIS IS DONE BY BFS!!!
; c603:     MOVE    (A5),D0
;           CALLA   T12
;           MOVE    (A5),D1
;           CALLA   T12
;           ADD     (A5),D1
;           CALLA   T12
;           ADD     (A5),D1
;           CMP     D0,D1           ; e1+e2+e3 > e0
;           JMPA    GT,nextcode
;*
;* Use the step starting at C611 to get a character index. If the index equals
;* either a foreward or reverse start/stop character set the foreward decode
;* flag accordingly and enter the decoding algorithm. Otherwise, quit (no
;* start charcter found). Then, test the character widths. If all tests pass
;* set last width equal to the sum of the character elements generated by step
;* C616 and store the character found, else quit.
;*
c604:       CALLA   c611
            CMP.B   #0,DL7
            JMPA    EQ,nextcode
            CMP     #4,DL7          ; "C"
            JMPR.S  EQ,c604_2
            CMP.B   #6,DL7          ; "D"
            JMPR.S  EQ,c604_2
            CMP.B   #10,DL7         ; "A"
            JMPR.S  EQ,c604_2
            CMP.B   #13,DL7         ; "B"
            JMPR.S  EQ,c604_2
            CMP.B   #14,DL7         ; "A"
            JMPR.S  EQ,c604_1
            CMP.B   #16,DL7         ; "D"
            JMPR.S  EQ,c604_1
            CMP.B   #20,DL7         ; "B"
            JMPR.S  EQ,c604_1
            CMP.B   #25,DL7         ; "C"
            JMPA    NE,nextcode
c604_1:     BCLR    #FOREWARD,SR
            JMPR.S  c604_3
c604_2:     BSET    #FOREWARD,SR
c604_3:     CLR     LABEL_BUF
            MOVE    #LABEL_BUF+1,A4
            CALLA   c615
            MOVE    D0,A0           ; store last width
c605:       CALLA   room
c606:       CALLA   c611            ; get char pattern & character
            CMP     #$78,DH7
            JMPA    EQ,nextcode
            CALLA   c615            ; check widths
            CMP     #0,DH7
            JMPA    EQ,nextcode
c607:       MOVE    A0,A1           ; ck   4/5 > CW/LW > 5/4
            LSR     #2,A1
```

```
        ADD     A0,A1
        CMP     A1,D0           ; CV > LV*5/4
        JMPA    GT,nextcode
        MOVE    D0,D1
        LSR     #2,D1
        ADD     D0,D1
        CMP     D1,A0           ; LV > CV*5/4
        JMPA    GT,nextcode
c608:   MOVE    IPTR,A5         ; ck  1.5 > CV/e0 > 30.0
        MOVE    (A5),D2
        MOVE    D2,D1
        ASL     #5,D1           ; e0*32
        SUB     D2,D1
        SUB     D2,D1
        CMP     D1,D0           ; CV > e0*30
        JMPA    GT,nextcode
        MOVE    D2,D1
        LSR     D1
        ADD     D2,D1
        CMP     D1,D0           ; CV < e0*3/2
        JMPA    LT,nextcode
c609:   CMP.B   #$41,DH7
        JMPR.S  EQ,c610
        CMP.B   #$42,DH7
        JMPR.S  EQ,c610
        CMP.B   #$43,DH7
        JMPR.S  EQ,c610
        CMP.B   #$44,DH7
        JMPR.S  EQ,c610
        MOVE.B  LABEL_BUF,DH7
        CMP.B   #36,DH7
        JMPA    LE,c605
c610:   MOVE    IPTR,A5
        ADD     #16,A5          ; point to e(i+8)
        CMP     #CODEND,A5
        JMPR.S  LT,c610_1
        SUB     #WIDTH,A5
c610_1: MOVE    (A5),D0         ; e(i+8)
        CALLA   TI_2
        MOVE    (A5),D1
        CALLA   TI_2
        ADD     (A5),D1
        CALL    TI_2
        ADD     (A5),D1         ; e7+e6+e5
        CMP     D0,D1           ; e7+e6+e5 > e(i+8)
        JMPA    GT,nextcode
        MOVE.B  LABEL_BUF,DL7
        MOVE.B  #SCB,DH7        ; send out label out
        CALLA   outch
        MOVE.B  DL7,DH7
        CALLA   outch           ; send the count
        BTST    #FOREWARD,SR
        JMPR.S  CC,c610_3
        MOVE    #LABEL_BUF,A0
c610_2: MOVE.B  (A0)+,DH7
        CALLA   outch
        DJNZ.B  DL7,c610_2
        JMPA    found_label
c610_3: MOVE.B  -(A4),DH7
        CALLA   outch
        DJNZ.B  DL7,c610_3
        JMPA    found_label
```

```
x125bs:  DC.B     'xxx7x40xx29x6xxxx18x5xxx3xxxxxxx'

code125: BTST     #125,DECODER1
         JMPA     CC,codeUPC
         MOVE     last1,A5
         MOVE     A5,IPTR
;*
;* If element(i) < min margin ratio * (element(i+1)+element(i+2)), then quit,
;* margin too small.
;*
                                    ; check min margin ratios
c503:    MOVE     (A5),D0           ; e0
         CALLA    T12
         MOVE     (A5),D1           ; e1
         CALLA    T12
         MOVE     (A5),D2           ; e2
         ADD      D1,D2
         MOVE     D2,D3
         ADD      D2,D2
         ADD      D3,D2             ; D2 = (e1+e2)*3
         CMP      D2,D0             ; e0 < (e1+e2)*3
         JMPA     LT,codeUPC

;*
;* Look for a valid start pattern. Check if element(i+1)/element(i+2) > max
;* narrow element ratio (3.0) or if element(i+2)/element(i+1) > max narrow
;* element ratio (3.0). If so, quit. Else, determine direction of scan.
;* Check if element(i+3)/element(i+1) > start-stop threshold (1.5). If so
;* set foreward decode flag true (start pattern has two narrow bars, backward
;* stop pattern has narrow bar followed by wide bar). If foreward, check if
;* element(i+2)/element(i+4) > min element ratio (1.5); if so, quit (two bars
;* are not equal width). If okay, check if element(i+1)/element(i+3) > min
;* element ratio (1.5). If so,quit. If foreward is false check if element
;* (i+3)/element(i+1) < max element ratio (5.0). If not so, quit. Otherwise,
;* set foreward decode flag false.
;* If okay to here, set last char width to the sum of elements(i+1) + element
;* (i+2) * margin scaler (8.0), set label string to empty, and increment IPTR
;* by 8.
;*                                   TEST MAX NARROW ELEMENT RATIO
c504:    MOVE     (A5),D2           ; e2
         MOVE     D2,D3
         ADD      D2,D3
         ADD      D2,D3
         CMP      D3,D1             ; e1/e2 > 3 ?
         JMPA     GT,codeUPC        ; REM D1 = e1
         MOVE     D1,D3
         ADD      D1,D3
         ADD      D1,D3
         CMP      D3,D2             ; e2/e1 > 3 ?
         JMPA     GT,codeUPC
         CALLA    T12               ; e3
         MOVE     (A5),D3           ; TEST DIRECTION OF SCAN
         MOVE     D1,D4
         LSR      #1,D4
         ADD      D1,D4
         CMP      D4,D3             ; e3/e1 > 1.5 (ift ==> foreward=true)
         JMPR.S   GT,c504_1         ; FOREWARD DECODE ?
         MOVE     D1,D4             ; BACKWARD DECODE !
         ASL      #2,D4
         ADD      D1,D4
         CMP      D3,D4             ; e3/e1 < 5.0
         JMPA     GT,codeUPC
         BCLR     #FOREWARD,SR
         CALLA    T12               ; move IPTR to IPTR+8
         JMPR.S   c504_2
c504_1:  CALLA    T12
         MOVE     (A5),D4           ; e4
         MOVE     D2,D5
         LSR      #1,D5
         ADD      D2,D5
         CMP      D5,D4             ; e4/e2 > 1.5
         JMPA     GT,codeUPC
         MOVE     D4,D5
         LSR      #1,D5
         ADD      D4,D5
         CMP      D5,D2             ; e2/e4 > 1.5
```

```
          JMPA     GT,codeUPC
          MOVE     D3,D5
          LSR      #1,D5
          ADD      D5,D3
          CMP      D3,D1          ; e1/e3 > 1.5
          JMPA     GT,codeUPC
          BSET     #FOREWARD,SR
c504_2:   MOVE     A5,IPTR        ; IPTR := IPTR+8
          ADD      D2,D1
          ASL      #3,D1
          MOVE     D1,A0          ; LAST_WIDTH := (e1+e2)*8
          CLR      LABEL_BUF
          MOVE     #LABEL_BUF+1,A4
c505:     CALLA    room
;*
;* Find the widest, narrowest and totals for the bars and spaces in the
;* current character. If #FOREWARD is true (bit set) then this loop
;* exits with:
;*               D1 = e1+e3+e5+e7+e9 (bar total)
;*               D3 = widest bar
;*               D5 = narrowest bar
;*               D2 = e2+e4+e6+e8+e10 (space total)
;*               D4 = widest space
;*               D6 = narrowest space
;* else, if #FOREWARD is false (bit clear) then:
;*               D1 = e0+e2+e4+e6+e8 (space total)
;*               D3 = widest space
;*               D5 = narrowest space
;*               D2 = e1+e3+e5+e7+e9 (bar total)
;*               D4 = widest bar
;*               D6 = narrowest bar
;*
c511:     MOVE     IPTR,A5
          BTST     #FOREWARD,SR
          JMPR.S   CC,c511_1
          CALLA    T12            ; if foreward decode e1 ==> D1
c511_1:   MOVE     (A5),D1        ; else               e0 ==> D1
          MOVE     D1,D3
          MOVE     D3,D5
          CALLA    T12            ; if foreward decode e2 ==> D2
          MOVE     (A5),D2        ; else               e1 ==> D2
          MOVE     D2,D4
          MOVE     D2,D6
          MOVE.B   #4,DL7
c511_2:   CALLA    T12
          CMP      (A5),D3        ; D3 > (A5)
          JMPR.S   GE,c511_3
          MOVE     (A5),D3
c511_3:   CMP      (A5),D5        ; D5 < (A5)
          JMPR.S   LE,c511_4
          MOVE     (A5),D5
c511_4:   ADD      (A5),D1
          CALLA    T12
          CMP      (A5),D4        ; D4 > (A5)
          JMPR.S   GE,c511_5
          MOVE     (A5),D4
c511_5:   CMP      (A5),D6        ; D6 < (A5)
          JMPR.S   LE,c511_6
          MOVE     (A5),D6
c511_6:   ADD      (A5),D2
          DJNZ.B   DL7,c511_2
          BTST     #FOREWARD,SR   ; if backward decode swap in wide/
          JMPR.S   CS,c511_7      ; narrow bars and spaces and totals
          EXG      D3,D4          ; to match.
          EXG      D5,D6
          EXG      D1,D2
                                  ; AT EXIT:
                                  ;    D1 = e1+e3+e5+e7+e9 (bar total)
                                  ;    D3 = widest bar
                                  ;    D5 = narrowest bar
                                  ;    D2 = e2+e4+e6+e8+e10 (space total)
                                  ;    D4 = widest space
                                  ;    D6 = narrowest space
c511_7:   MOVE     D1,D0
          ADD      D2,D0          ; calc character width
          MOVE     D1,A1
          ASL      #3,A1
          SUB      D1,A1          ; A1 = bar thresh
          LSR      #5,A1          ; calc bar thresh    7/32*tn
          MOVE     D2,A2
```

```
        ASL     #3,A2
        SUB     D2,A2               ; A2 = space thresh
        LSR     #5,A2               ; calc space thesh     7/32*tn
;*
;* Set a pair of binary numbers, representing the bar and space patterns to
;* zero. If foreward is true, use elements in order e1, e3, e5, e7 and e9;
;* if foreward is false, use elements in order e9, e7, e5, e3, and e1. These
;* ordered elements are compared with the bar threshold. If the element under
;* consideration is greater than the calculated threshold the binary bar
;* pattern is multiplied by 2 and has 1 added to it, otherwise it is multi-
;* plied by 2 only. Similarly, for the space pattern ordered elements e2, e4,
;* e6, e8, and e10 are considered if foreward is true, otherwise ordered
;* elements e8, e6, e4, e2, and e0 are considered.
;*
c512a:  CLR     D1
        CLR     D2
        MOVE.B  #5,DL7
        BTST    #FOREWARD,SR        ; REM if foreward = false
        JMPR.S  CC,c512b_1          ; IPTR ==> e9
;*
        MOVE    IPTR,A5             ; IF FOREWARD = true
c512a_1: CALLA  TI2                 ; starting at e1, calc bar pattern
        ADD     D1,D1               ; looking at e1,e3,e5,e7,e9
        CMP     (A5),A1
        JMPR.S  LE,c513a
        ADD     #1,D1
c513a:  CALLA   TI2                 ; starting at e2, calc space pattern
        ADD     D2,D2               ; looking at e2,e4,e6,e8,e10
        CMP     (A5),D2
        JMPR.S  LE,c513a_1
        ADD     #1,D2
c513a_1: DJNZ.B DL7,c512a_1
        JMPR.S  c514
;*                                  ; IF FOREWARD = false
c512b_1: ADD    D1,D1               ; starting at e9, calc bar pattern
        CMP     (A5),A1             ; looking at e9,e7,e5,e3,e1
        JMPR.S  LE,c513b
        ADD     #1,D1
c513b:  CALLA   TI_2
        ADD     D2,D2               ; starting at e8, calc space pattern
        CMP     (A5),A2             ; looking at e8,e6,e4,e2,e0
        JMPR.S  LE,c513b_1
        ADD     #1,D2
c513b_1: CALLA  TI_2
        DJNZ.B  DL7,c512b_1
;*
;* Use the binary patterns calculated as a pointer to select a character
;* from the pattern xi25bs, indexed at 0. If the character pattern is 'x'
;* indicate bad pattern and return, else get character pattern.
;*
c514:   MOVE    #xi25bs,A1
        MOVE    A1,A2
        ADD     D1,A1
        MOVE.B  (A1),DL1
        CMP.B   #'x',DL1            ; if char 'x', quit
        JMPA    EQ,c510
        ADD     D2,A2
        MOVE.B  (A2),DL2
        CMP.B   #'x',DL2
        JMPA    EQ,c510
                                    ; AT EXIT  D1 ==> bar character
                                    ;          D2 ==> space character
                                    ;          D0 ==> char width sum
                                    ;          D3 = widest bar
                                    ;          D5 = narrowest bar
                                    ;          D4 = widest space
                                    ;          D6 = narrowest space
;*
;* Check element widths. Ref 5.3.15 to 5.3.21.
;*
c516:   MOVE    D6,D7
        ASL     #2,D7
        ADD     D6,D7
        CMP     D7,D4               ; widest_sp/narrowest_sp > 5
        JMPA    GT,c510
        MOVE    D6,D7
        LSR     #1,D7
        ADD     D6,D7
        CMP     D7,D4               ; widest_sp/narrowest_sp < 1.5
        JMPA    LT,c510
```

```
c517:   MOVE    D5,D7
        ASL     #2,D7
        ADD     D5,D7
        CMP     D7,D3           ; widest_bar/narrowest_bar > 5
        JMPA    GT,c510
c518:   MOVE    D5,D7
        ASL     #1,D7
        ADD     D5,D7
        CMP     D7,D3           ; widest_bar/narrowest_bar < 1.5
        JMPA    LT,c510
c519:   MOVE    D6,D7
        ADD     D6,D7
        ADD     D6,D7
        CMP     D5,D7           ; narrowest_bar/narrowest_sp > 3
        JMPA    GT,c510
c520:   MOVE    D5,D7
        ADD     D5,D7
        ADD     D5,D7
        CMP     D6,D7           ; narrowest_sp/narrowest_bar > 3
        JMPA    GT,c510
;*
;* Compute the ratio of the sum of the elements in the current frame width to
;* the last frame width. If this is the first character pair (start/stop),
;* then check if this ratio > max margin char ratio (2.0) or if 1/ratio is
;* > max margin char ratio. If it is, then quit (no char found). If not the
;* first time through, then if this ratio is greater than max char ratio
;* (1.25) or less than min char ratio (.80), then go to step 5.3.9.
;*
c507:   MOVE.B  LABEL_BUF,DH7
        CMP.B   #0,DH7
        JMPR.S  NE,c507_1       ; FIRST CHARACTER
        MOVE    A0,D4           ; get Last char Width
        ADD     D4,D4
        CMP     D4,D0           ; CW/LW > 2
        JMPA    GT,codeUPC
        MOVE    D0,D4
        ADD     D4,D4
        CMP     D4,A0           ; LW/CW > 2
        JMPA    GT,codeUPC
        JMPA    c508
c507_1: MOVE    A0,D4           ; 1+N CHARACTER
        LSR     #2,D4
        ADD     A0,D4
        CMP     D4,D0           ; CW/LW > 5/4 ==> CW > LW*1.25
        JMPA    GT,c510
        MOVE    D0,D4
        LSR     #2,D4
        ADD     D0,D4
        CMP     D4,A0           ; CW/LW < 4/5 ==> LW > CW*1.25
        JMPA    GT,c510
;*
;* If the length of the label string is is the maximum allowable label
;* length, quit! Otherwise, set LAST_WIDTH to current CHAR_WIDTH, add
;* frame_width (20) to IPTR and append the decoded characters to the
;* end of the label string. If foreward decode is true append bar char
;* and then space char. If foreward is false append space char and, then,
;* bar character to the label string. Then, re-enter the decoding loop.
;*
c508:   MOVE.B  I25LL,DL7       ; get max allowable I25 label length
        CMP.B   DH7,DL7         ; maxLL > DH7 ( actual label length)
        JMPA    GE,codeUPC
        MOVE    D0,A0           ; LAST_WIDTH := CURRENT_CHAR_WIDTH
        MOVE    IPTR,A5
        ADD     #20,A5          ; IPTR := IPTR+20
        CMP     #CDEND,A5
        JMPR.S  LT,c508_1
        SUB     #WIDTH,A5
c508_1: MOVE    A5,IPTR
        ADD.B   #2,LABEL_BUF    ; add decoded char's
        BTST    #FOREWARD,SR    ; IF FOREWARD = true
        JMPR.S  CC,c508_2       ; append bar char + space char
        MOVE.B  DL1,(A4)+
        MOVE.B  DL2,(A4)+
        JMPA    c505
c508_2: MOVE.B  DL2,(A4)+       ; IF FOREWARD = false
        MOVE.B  DL1,(A4)+       ; append space char + bar char
        JMPA    c505            ; re-enter decoding loop at top
;*
;* Check for possible end of label.
;* Check if element(i+4)/(element(i+2)+element(i+3)) > min margin ratio (3.0).
```

```
;* If not, quit. If okay, check if element(i+3)/element(i+2) > max narrow
;* element ratio (3.0), or if element(i+2)/element(i+3) > max narrow element
;* ratio (3.0). If so, quit. If okay, check that (element(i+2)+element(i+3)
;* times margin scalar (8.0)/last char width > max margin char ratio (2.0) or
;* < 1/max margin char ratio (0.5). If so, quit.
;* Then, if foreward is true check that element(i+1)/element(i+3) is greater
;* than max element ratio (5.0) or less than min element ratio (1.5). If so,
;* quit. If okay, send the string out.
;* If foreward is false check that element(i+1)/element(i+3) is greater than
;* the start-stop ratio (1.5) or is less than 1/min element ratio (0.6667).
;* If so, quit. Then, check that element(i)/element(i+2) is greater than
;* min element ratio (1.5) or less than 1/min element ratio (0.6667). If so,
;* quit. If okay, reverse the order of the string and send it out.
;*
c510:     MOVE      #LABEL_BUF,A2     ; get label base address
          MOVE.B    (A2),DL7          ; get label length
          CMP.B     #0,DL7
          JMPA      EQ,codeUPC
;*
          MOVE      IPTR,A5
          MOVE      (A5),D0           ; get e0
          CALLA     T12
          MOVE      (A5),D1           ; e1
          CALLA     T12
          MOVE      (A5),D2           ; e2
          CALLA     T12
          MOVE      (A5),D3           ; e3
          CALLA     T12               ; move pointer to e4
          MOVE      D2,D4             ; ck   e4/e2+e3 > 3.0
          ADD       D3,D4             ; D4 = e2+e3
          MOVE      D4,D5
          ADD       D4,D5
          ADD       D4,D5             ; D5 = 3*(e2+e3)
          CMP       (A5),D5           ; 3*(e2+e3) < e4
          JMPA      LT,codeUPC
          MOVE      D2,D5             ; e2
          ADD       D5,D5
          ADD       D2,D5             ; e2*3
          CMP       D5,D3             ; e3/e2 > 3.0
          JMPA      GT,codeUPC
          MOVE      D3,D5             ; e3
          ADD       D3,D5
          ADD       D3,D5             ; e3*3
          CMP       D5,D2             ; e2/e3 > 3.0
          JMPA      GT,codeUPC
          MOVE      D4,D5             ; ck  1/2 > (e2+e3)*8/LW > 2
          ASL       #3,D5             ; (e2+e3)*8       REM   D4 = e2+e3
          MOVE      A0,A1
          ADD       A1,A1             ; LW*2
          CMP       A1,D5             ;            (e2+e3)*8 > 2*LW
          JMPA      GT,codeUPC
          MOVE      A0,A1             ;     1/2*LW > (e2+e3)*8
          LSR       #1,A1
          CMP       D5,A1
          JMPA      GT,codeUPC
;*
          BTST      #FOREWARD,SR      ; IF FOREWARD = true
          JMPR      CC,c510_2         ; ck    1.5 > e1/e3 > 5.0
;*
          MOVE      D3,D5             ; e3
          ASL       #2,D5             ; e3*4
          ADD       D3,D5             ; e3*5
          CMP       D5,D1             ; e1 > e3*5
          JMPA      GT,codeUPC
          MOVE      D3,D5             ; e3
          LSR       #1,D5             ; e3/2
          ADD       D3,D5             ; e3/2 + e3 = e3*3/2
          CMP       D5,D1             ; e1 < e3*3/2
          JMPA      LT,codeUPC
          MOVE.B    #$25,DH7          ; send label type
          CALLA     outch
          ADD.B     #1,DL7            ; send label length and the label
c510_1:   MOVE.B    (A2)+,DH7
          CALLA     outch
          DJNZ.B    DL7,c510_1
          JMPA      found_label
;*
c510_2:   MOVE      D3,D5             ; ck    2/3 > e1/e3 > 3/2
          LSR       #1,D5             ;            e1/e3 > 3/2  ==> e1 > e3*3/2
          ADD       D3,D5
```

```
         CMP      D5,D1
         JMPA     GT,codeUPC
         MOVE     D1,D5        ;        2/3 > e1/e3          ==> e3 > e1*3/2
         LSR      #1,D5
         ADD      D1,D5
         CMP      D5,D3
         JMPA     LT,codeUPC
         MOVE     D2,D5        ; ck   2/3 > e0/e2 > 3/2
         LSR      #1,D5        ;            e0/e2 > 3/2
         ADD      D2,D5
         CMP      D5,D0
         JMPA     GT,codeUPC
         MOVE     D0,D5        ;        2/3 > e0/e2          ==> e2 > e0*3/2
         LSR      #1,D5
         ADD      D0,D5
         CMP      D5,D4
         JMPA     LT,codeUPC
         MOVE.B   #$25,DH7     ; send label type
         CALLA    outch
         MOVE.B   DL7,DH7      ; send label length
         CALLA    outch
c510_3:  MOVE.B   -(A6),DH7
         CALLA    outch
         DJNZ.B   DL7,c510_3
         JMPA     found_label
```

```
xc93tbl:  DC.B      '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ-. $/+%&'
xc93ec:   DC.B      $1B,$1C,$1D,$1E,$1F
          DC.B      ';<=>?[\]^_{|}~'
          DC.B      $7F,0,$40,$60

;*
;* If foreward is true, sum elements e1 through e6. Then calculate the four
;* sum terms T1=e1+e2, T2=e2+e3, T3=e3+e4, T4=e4+e5. If foreward is false,
;* sum elements e2 through e7. Then calculate the four sum terms T1=e6+e7,
;* T2=e5+e6, T3=e4+e5, T4=e3+e4. Reference sections 7.3.12 through 7.3.14
;* of the technical documention.
;*
c712:     MOVE      IPTR,A5
          BTST      #FOREWARD,SR
          JMPR.S    CC,c712b
c712a:    CALLA     T12             ; e1
          MOVE      (A5),D1
          MOVE      D1,D0
          CALLA     T12             ; e2
          MOVE      (A5),D2
          ADD       D2,D0
          ADD       D2,D1           ; T1 = e1+e2
          CALLA     T12             ; e3
          MOVE      (A5),D3
          ADD       D3,D0
          ADD       D3,D2           ; T2 = e2+e3
          CALLA     T12             ; e4
          MOVE      (A5),D4
          ADD       D4,D0
          ADD       D4,D3           ; T3 = e3+e4
          CALLA     T12             ; e5
          ADD       (A5),D0
          ADD       (A5),D4         ; T4 = e4+e5
          CALLA     T12             ; e6
          ADD       (A5),D0         ; D0 = e1+e2+e3+e4+e5+e6
          JMPR.S    c715
c712b:    CALLA     T14             ; e2
          MOVE      (A5),D0
          CALLA     T12             ; e3
          MOVE      (A5),D4
          ADD       D4,D0
          CALLA     T12             ; e4
          MOVE      (A5),D3
          ADD       D3,D0
          ADD       D3,D4           ; T4 = e3+e4
          CALLA     T12             ; e5
          MOVE      (A5),D2
          ADD       D2,D0
          ADD       D2,D3           ; T3 = e4+e5
          CALLA     T12             ; e6
          MOVE      (A5),D1
          ADD       D1,D0
          ADD       D1,D2           ; T2 = e5+e6
          CALLA     T12             ; e7
          ADD       (A5),D1         ; T1 = e6+e7
          ADD       (A5),D0         ; D0 = e2+e3+e4+e5+e6+e7
;*
;* Compute the three threshold values by multiplying the character width
;* calculated above by the three threshold values 2.5/9, 3.5/9, and 4.5/9
;*
c715:     CLR       A0
          MOVE      D0,A1
          MOVE      #9,D7
          DIVU      A2,A0           ; A0 = CW/9
          MOVE      A0,A1
          LSR       #1,A1
          ADD       A0,A1
          ADD       A0,A1           ; A1 = CW/9 * 2.5  ==> thresh1
          MOVE      A1,A2
          ADD       A0,A2           ; A2 = CW/9 * 3.5  ==> thresh2
          MOVE      A2,A3
          ADD       A0,A3           ; A3 = CW/9 * 4.5  ==> thresh3
;*
;* Compute the four digits of the character pattern, (d)1 thru (d)4, by doing
;* the following for each sum T1 through T4. For j 1 through 4:
;*              (d)j = 2 if Tj < thresh1
```

```
;*              (d)j = 3 if Tj < thresh2
;*              (d)j = 4 if Tj < thresh3
;*              (d)j = 5 if Tj >= thresh4
;* The pattern is equal to:
;*              ((d)4) + 16*((d)3) + 256*((d)2) + 4096*((d)1)
;*
c716:     CMP      A1,D1           ; T1 < thresh1
          JMPR.S   GE,c716_1
          MOVE     #$2000,D5
          JMPR.S   c716_4
c716_1:   CMP      A2,D1           ; T1 < thresh2
          JMPR.S   GE,c716_2
          MOVE     #$3000,D5
          JMPR.S   c716_4
c716_2:   CMP      A3,D1           ; T1 < thresh3
          JMPR.S   GE,c716_3
          MOVE     #$4000,D5
          JMPR.S   c716_4
c716_3:   MOVE     #$5000,D5       ; T1 >= thresh3
c716_4:   CMP      A1,D2           ; T2 < thresh1
          JMPR.S   GE,c716_5
          ADD      #$0200,D5
          JMPR.S   c716_8
c716_5:   CMP      A2,D2           ; T2 < thresh2
          JMPR.S   GE,c716_6
          ADD      #$0300,D5
          JMPR.S   c716_8
c716_6:   CMP      A3,D2           ; T2 < thresh3
          JMPR.S   GE,c716_7
          ADD      #$0400,D5
          JMPR.S   c716_8
c716_7:   ADD      #$0500,D5       ; T2 >= thresh3
c716_8:   CMP      A1,D3           ; T3 < thresh1
          JMPR.S   GE,c716_9
          ADD      #$0020,D5
          JMPR.S   c716_12
c716_9:   CMP      A2,D3           ; T3 < thresh2
          JMPR.S   GE,c716_10
          ADD      #$0030,D5
          JMPR.S   c716_12
c716_10:  CMP      A3,D3           ; T3 < thresh3
          JMPR.S   GE,c716_11
          ADD      #$0040,D5
          JMPR.S   c716_12
c716_11:  ADD      #$0050,D5       ; T3 >= thresh3
c716_12:  CMP      A1,D4           ; T4 < thresh1
          JMPR.S   GE,c716_13
          ADD      #$0002,D5
          RET
c716_13:  CMP      A2,D4           ; T4 < thresh2
          JMPR.S   GE,c716_14
          ADD      #$0003,D5
          RET
c716_14:  CMP      A3,D4           ; T4 < thresh3
          JMPR.S   GE,c716_15
          ADD      #$0004,D5
          RET
c716_15:  ADD      #$0005,D5       ; T4 >= thresh3
          RET                      ; AT EXIT  D0 ==> char width
                                   ;          D5 ==> char pattern


codeC93:  BTST     #C93,DECODER1
          JMPA     CC,nextcode
          MOVE     lastI,A5
          MOVE     A5,IPTR
;*
;* Check that element(I) > 3*(element(I+1) + element(I+2)).
;*
c703:     MOVE     (A5),D0
          CALLA    T12
          MOVE     (A5),D1
          CALLA    T12
          ADD      (A5),D1
          MOVE     D1,D2
          ADD      D1,D1
          ADD      D2,D1
```

```
         CMP      D1,D0              ; e0 < 3*(e1+e2)
         JMPA     LE,nextcode
c704:    BSET     #!FOREWARD,SR
         CALLA    c712               ; AT EXIT  D0 ==> char width
                                     ;          D5 ==> char pattern
         CMP      #$2225,D5          ; FOREWARD START ?
         JMPR.S   NE,c704_1
         BSET     #FOREWARD,SR
         JMPR.S   c704_2
c704_1:  CMP      #$2552,D5          ; BACKWARD START ?
         JMPA     NE,nextcode
         MOVE     IPTR,A5
         CALLA    TI2
         MOVE     (A5),D1
         CALLA    TI4
         MOVE     (A5),D3
         MOVE     D1,D2
         ADD      D2,D2
         CMP      D2,D3              ; e3/e1 > 2.0
         JMPA     GT,nextcode
         ADD      D3,D3
         CMP      D3,D1              ; e1/e3 > 2.0
         JMPA     GT,nextcode
         BCLR     #FOREWARD,SR
c704_2:  MOVE     IPTR,A5
         CALLA    TI2
         BTST     #FOREWARD,SR
         JMPR.S   CC,c704_3
         CALLA    TI2
c704_3:  MOVE     (A5),D1
         MOVE     D1,D3
         CALLA    TI2
         MOVE     (A5),D2
         MOVE     D2,D4
         MOVE.B   #2,DL7
;*
c704_4:  CALLA    TI2
         CMP      (A5),D1            ; e3 or e4 > (A5)
         JMPR.S   GE,c704_5
         MOVE     (A5),D1
c704_5:  CMP      (A5),D3            ; e3 or e4 < (A5)
         JMPR.S   LE,c704_6
         MOVE     (A5),D3
c704_6:  CALLA    TI2
         CMP      (A5),D2            ; e4 or e5 > (A5)
         JMPR.S   GE,c704_7
         MOVE     (A5),D2
c704_7:  CMP      (A5),D4            ; e4 or e5 < (A5)
         JMPR.S   LE,c704_8
         MOVE     (A5),D4
c704_8:  DJNZ.B   DL7,c704_4
         BTST     #FOREWARD,SR       ; AT EXIT  D1 ==> wide bar
         JMPR.S   CS,c704_9          ;          D2 ==> wide space
         EXG      D1,D2              ;          D3 ==> narrow bar
         EXG      D3,D4              ;          D4 ==> narrow space
c704_9:  ASL      #3,D3              ; ck       wb/nb > 8.0
         CMP      D3,D1
         JMPA     GT,nextcode
         ASL      #3,D4              ; ck       ws/ns > 8.0
         CMP      D4,D2
         JMPA     GT,nextcode
         CLR      LABEL_BUF
         MOVE     #LABEL_BUF+1,A4
         MOVE     D0,LAST_WIDTH
;*
charloop: MOVE    IPTR,A5
         ADD      #12,A5             ; increment pointer by 1 frame width
         CMP      #GDEND,A5
         JMPR.S   LT,charl1
         SUB      #WIDTH,A5
charl1:  MOVE     IPTR,A5
         CALLA    c712               ; AT EXIT  D0 ==> char width
                                     ;          D5 ==> char pattern
         CMP.B    #$22,DH5
         JMPR.S   NE,c706_8
         CMP.B    #$22,DL5
         JMPR.S   NE,c706_1
         MOVE.B   #'7',DH7           ; 2222 = '7'
         JMPR.S   c707
c706_1:  CMP.B    #$23,DL5
```

```
                JMPR.S     NE,c706_2
                MOVE.B     #'L',DN7           ; 2223 = 'L'
                JMPR.S     c707
    c706_2:     CMP.B      #$25,DL5
                JMPR.S     NE,c706_2
                MOVE.B     #'(',DN7           ; 2225 = '('     ==> FWD START
                JMPR.S     c707
    c706_3:     CMP.B      #$33,DL5
                JMPR.S     NE,c706_4
                MOVE.B     #'1',DN7           ; 2233 = '1'
                JMPR.S     c707
    c706_4:     CMP.B      #$34,DL5
                JMPR.S     NE,c706_5
                MOVE.B     #'M',DN7           ; 2234 = 'M'
                JMPR.S     c707
    c706_5:     CMP.B      #$44,DL5
                JMPR.S     NE,c706_6
                MOVE.B     #'2',DN7           ; 2244 = '2'
                JMPR.S     c707
    c706_6:     CMP.B      #$45,DL5
                JMPR.S     NE,c706_7
                MOVE.B     #'N',DN7           ; 2245 = 'N'
                JMPR.S     c707
    c706_7:     CMP.B      #$55,DL5
                JMPA       NE,nextcode
                MOVE.B     #'3',DN7           ; 2255 = '3'
                JMPR.S     c707
    c706_8:     CMP.B      #23,DH5
                JMPR.S     NE,c706_14
                CMP.B      #$32,DL5
                JMPR.S     NE,c706_9
                MOVE.B     #'G',DN7           ; 2332 = 'G'
                JMPR.S     c707
    c706_9:     CMP.B      #$33,DL5
                JMPR.S     NE,c706_10
                MOVE.B     #'U',DN7           ; 2333 = 'U'
                JMPR.S     c707
    c706_10:    CMP.B      #$34,DL5
                JMPR.S     NE,c706_11
                MOVE.B     #'/',DN7           ; 2334 = '/'
                JMPR.S     c707
    c706_11:    CMP.B      #$43,DL5
                JMPR.S     NE,c706_12
                MOVE.B     #'H',DN7           ; 2343 = 'H'
                JMPR.S     c707
    c706_12:    CMP.B      #$44,DL5
                JMPR.S     NE,c706_13
                MOVE.B     #'X',DN7           ; 2344 = 'X'
    c706_13:    CMP.B      #$54,DL5
                JMPA       NE,nextcode
                MOVE.B     #'I',DN7           ; 2354 = 'I'
                JMPR.S     c707
    c706_14:    CMP.B      #$24,DH5
                JMPR.S     NE,c706_15
                CMP.B      #$43,DL5
                JMPA       NE,nextcode
                MOVE.B     #'+',DH7           ; 2443 = '+'
                JMPR.S     c707
    c706_15:    CMP.B      #$25,DH5
                JMPR.S     NE,c706_16
                CMP.B      #$52,DL5
                JMPA       NE,nextcode
                MOVE.B     #')',DN7           ; 2552 = ')'     ==> BKWD START
                JMPR.S     c707
    c706_16:    CMP.B      #$32,DH5
                JMPR.S     NE,c706_22
                CMP.B      #$22,DL5
                JMPR.S     NE,c706_17
                MOVE.B     #'A',DN7           ; 3222 = 'A'
                JMPR.S     c707
    c706_17:    CMP.B      #$23,DL5
                JMPR.S     NE,c706_18
                MOVE.B     #'S',DN7           ; 3223 = 'S'
                JMPR.S     c707
    c706_18:    CMP.B      #$24,DL5
                JMPR.S     NE,c706_19
                MOVE.B     #'Z',DN7           ; 3224 = 'Z'
                JMPR.S     c707
    c706_19:    CMP.B      #$33,DL5
                JMPR.S     NE,c706_20
```

```
          MOVE.B    #'B',DH7          ; 3233 = 'B'
          JMPR.S    c707
c706_20:  CMP.B     #$34,DL5
          JMPR.S    NE,c706_21
          MOVE.B    #'T',DH7          ; 3234 = 'T'
          JMPR.S    c707
c706_21:  CMP.B     #$44,DL5
          JMPA      NE,nextcode
          MOVE.B    #'C',DH7          ; 3244 = 'C'
          JMPR.S    c707
c706_22:  CMP.B     #$33,DH5
          JMPR.S    NE,c706_30
          CMP.B     #$22,DL5
          JMPR.S    NE,c706_23
          MOVE.B    #'4',DH7          ; 3322 = '4'
          JMPR.S    c707
c706_23:  CMP.B     #$23,DL5
          JMPR.S    NE,c706_24
          MOVE.B    #'0',DH7          ; 3323 = '0'
          JMPR.S    c707
c706_24:  CMP.B     #$24,DL5
          JMPR.S    NE,c706_25
          MOVE.B    #'-',DH7          ; 3224 = '-'
          JMPR.S    c707
c706_25:  CMP.B     #$32,DL5
          JMPR.S    NE,c706_26
          MOVE.B    #'Q',DH7          ; 3232 = 'Q'
          JMPR.S    c707
c706_26:  CMP.B     #$33,DL5
          JMPR.S    NE,c706_27
          MOVE.B    #'5',DH7          ; 3233 = '5'
          JMPR.S    c707
c706_27:  CMP.B     #$34,DL5
          JMPR.S    NE,c706_28
          MOVE.B    #'[',DH7          ; 3234 = '['    CHAR ==> ^S
          JMPR.S    c707
c706_28:  CMP.B     #$43,DL5
          JMPR.S    NE,c706_29
          MOVE.B    #'R',DH7          ; 3243 = 'R'
          JMPR.S    c707
c706_29:  CMP.B     #$44,DL5
          JMPA      NE,nextcode
          MOVE.B    #'6',DH7          ; 3244 = '6'
          JMPA      c707
c706_30:  CMP.B     #$34,DH5
          JMPR.S    NE,c706_33
          CMP.B     #$32,DL5
          JMPR.S    NE,c706_31
          MOVE.B    #'J',DH7          ; 3432 = 'J'
          JMPR.S    c707
c706_31:  CMP.B     #$33,DL5
          JMPR.S    NE,c706_32
          MOVE.B    #'Y',DH7          ; 3433 = 'Y'
          JMPR.S    c707
c706_32:  CMP.B     #$43,DL5
          JMPA      NE,nextcode
          MOVE.B    #'&',DH7          ; 3243 = '&'    CHAR ==> ^+
          JMPR.S    c707
c706_33:  CMP.B     #$35,DH5
          JMPR.S    NE,c706_34
          CMP.B     #$42,DL5
          JMPA      NE,nextcode
          MOVE      #'2',DH7          ; 3542 = '2'
          JMPR.S    c707
c706_34:  CMP.B     #$42,DH5
          JMPR.S    NE,c706_37
          CMP.B     #$22,DL5
          JMPR.S    NE,c706_34
          MOVE.B    #'.',DH7          ; 4222 = '.'
          JMPR.S    c707
c706_35:  CMP.B     #$23,DL5
          JMPR.S    NE,c706_36
          MOVE.B    #'#',DH7          ; 4223 = '#'    CHAR ==> ^/
          JMPR.S    c707
c706_36:  CMP.B     #$33,DL5
          JMPA      NE,nextcode
          MOVE.B    #' ',DH7          ; 4233 = ' ' (space)
          JMPR.S    c707
c706_37:  CMP.B     #$43,DH5
          JMPR.S    NE,c706_41
```

```
                CMP.B    #$22,DL5
                JMPR.S   NE,c706_38
                MOVE.B   #'D',DH7        ; 4322 = 'D'
                JMPR.S   c707
c706_38:        CMP.B    #$23,DL5
                JMPR.S   NE,c706_39
                MOVE.B   #'U',DH7        ; 4323 = 'U'
                JMPR.S   c707
c706_39:        CMP.B    #$32,DL5
                JMPR.S   NE,c706_40
                MOVE.B   #'D',DH7        ; 4332 = 'D'    CHAR ==> ^Z
                JMPR.S   c707
c706_40:        CMP.B    #$33,DL5
                JMPA     NE,nextcode
                MOVE.B   #'E',DH7        ; 4333 = 'E'
                JMPR.S   c707
c706_41:        CMP.B    #$44,DH5
                JMPR.S   NE,c706_45
                CMP.B    #$22,DL5
                JMPR.S   NE,c706_42
                MOVE.B   #'O',DH7        ; 4422 = 'O'
                JMPR.S   c707
c706_42:        CMP.B    #$23,DL5
                JMPR.S   NE,c706_43
                MOVE.B   #'P',DH7        ; 4423 = 'P'
                JMPR.S   c707
c706_43:        CMP.B    #$32,DL5
                JMPR.S   NE,c706_44
                MOVE.B   #'V',DH7        ; 4432 = 'V'
                JMPR.S   c707
c706_44:        CMP.B    #$33,DL5
                JMPA     NE,nextcode
                MOVE.B   #'B',DH7        ; 4333 = 'B'
                JMPR.S   c707
c706_45:        CMP.B    #$45,DH5
                JMPR.S   NE,c706_46
                CMP.B    #$32,DL5
                JMPA     NE,nextcode
                MOVE.B   #'K',DH7        ; 4532 = 'K'
                JMPR.S   c707
c706_46:        CMP.B    #$53,DH5
                JMPR.S   NE,c706_47
                CMP.B    #$22,DL5
                JMPA     NE,nextcode
                MOVE.B   #'S',DH7        ; 5322 = 'S'
                JMPR.S   c707
c706_47:        CMP.B    #$54,DH5
                JMPR.S   NE,c706_48
                CMP.B    #$22,DL5
                JMPA     NE,nextcode
                MOVE.B   #'F',DH7        ; 5422 = 'F'
                JMPR.S   c707
c706_48:        CMP.B    #$55,DH5
                JMPA     NE,nextcode
                CMP.B    #$22,DL5
                JMPA     NE,nextcode
                MOVE.B   #'9',DH7        ; 5522 = '9'
c707:           MOVE     IPTR,A5
                CALLA    T12
                BTST     #FOREWARD,SR
                JMPR.S   CC,c717_1
                CALLA    T12
c717_1:         MOVE     (A5),D1
                MOVE     D1,D3
                CALLA    T12
                MOVE     (A5),D2
                MOVE     D2,D4
                MOVE.B   #2,DL7
;*
c717_2:         CALLA    T12
                CMP      (A5),D1         ; e3 or e4 > (A5)
                JMPR.S   GE,c717_3
                MOVE     (A5),D1
c717_3:         CMP      (A5),D3         ; e3 or e4 < (A5)
                JMPR.S   LE,c717_4
                MOVE     (A5),D3
c717_4:         CALLA    T12
                CMP      (A5),D2         ; e4 or e5 > (A5)
                JMPR.S   GE,c717_5
                MOVE     (A5),D2
```

```
c717_5:  CMP      (A5),D4          ; e4 or e5 < (A5)
         JMPR.S   LE,c717_6
         MOVE     (A5),D4
c717_6:  DJNZ.B   DL7,c717_2
         BTST     #FOREWARD,SR     ; AT EXIT  D1 ==> wide bar
         JMPR.S   CS,c718          ;          D2 ==> wide space
         EXG      D1,D2            ;          D3 ==> narrow bar
         EXG      D3,D4            ;          D4 ==> narrow space
c718:    ASL      #3,D3            ; ck       wb/nb > 8.0
         CMP      D3,D1
         JMPA     GT,nextcode
c719:    ASL      #3,D4            ; ck       ws/ns > 8.0
         CMP      D4,D2
         JMPA     GT,nextcode
c708:    MOVE     LAST_WIDTH,D1    ; ck       4/5 > CW/LW > 5/4
         MOVE     D1,D2
         LSR      #2,D2
         ADD      D1,D2
         CMP      D2,D0            ; CW >.LW*5/4
         JMPA     GT,nextcode
         MOVE     D0,D2
         LSR      #2,D2
         ADD      D0,D2
         CMP      D2,D1            ; LW > CW*5/4
         JMPA     GT,nextcode
c709:    CMP.B    #'(',DH7
         JMPR.S   EQ,c711
         CMP.B    #')',DH7
         JMPR.S   EQ,c711
c710:    MOVE     DH7,(A4)+
         MOVE     D0,LAST_WIDTH
         MOVE.B   LABEL_BUF,DL7
         CMP.B    #38,DL7          ; DL7 < 38
         JMPA     LT,cbarloop
         JMPA     nextcode

;*
;*
;*
c711:    MOVE     IPTR,A5
         SUB      #16,A5           ; decrement pointer to ????????
         CMP      A5,#GDATA
         JMPR.S   LT,c711_1
         ADD      #WIDTH,A5
c711_1:  MOVE     (A5),D0          ; e8
         CALLA    TI_2
         MOVE     (A5),D1          ; e7
         CALLA    TI_2
         MOVE     (A5),D2          ; e6
         ADD      D1,D2
         MOVE     D2,D3
         ADD      D3,D2
         ADD      D3,D2            ; D2 = 3*(e7+e6)
         CMP      D0,D2            ; 3*(e7+e6) > e8
         JMPA     GT,nextcode
         BTST     #FOREWARD,SR
         JMPR     CC,c711_2
         CALLA    TI_2
         MOVE     (A5),D2          ; D1 = e7, D2 = e5
         MOVE     D1,D3
         ADD      D3,D3
         CMP      D3,D2            ; e5/e7 > 2.0
         JMPA     GT,nextcode
         MOVE     D2,D3
         ADD      D3,D3
         CMP      D3,D1            ; e7/e5 > 2.0
         JMPA     GT,nextcode

c711_2:
```

```
;*
vtable:   DC.B      6,6,6,4,4,4,4,4,4,4,4,4,6,6,6,6,6,6,6,6
          DC.B      6,6,6,8,6,6,6,6,6,6,6,6,6,4,4,4,4,4,4,4
          DC.B      4,4,6,6,6,6,6,6,8,6,6,6,6,8,6,6,6,6,6,6
          DC.B      8,4,6,4,4,4,4,4,4,4,4,4,4,4,4,4,4,8,4,6
          DC.B      6,6,6,6,6,6,6,6,6,8,8,8,6,6,6,6,6,6,6,8
          DC.B      8,8,8,4,4,6,6,6
;*
;* Check for parity error using the character value found as an index into
;* VTABLE. Let the value found be Vn, if Vn+1.75/11 < BAR TOTAL or if
;* Vn-1.75/11 > BAR TOTAL, quit (parity error)
;*
c807:     MOVE      D0,D3
          LSR       #2,D3           ; CW/4
          MOVE      D3,D2
          ADD       D3,D3
          ADD       D2,D3           ; CW*3/4
          ADD       D0,D3           ; CW*1.75
          CLR       D2
          MOVE      #11,A5
          DIVU      A5,D2           ; D2 ==> CW*1.75/11
;*
          MOVE      #vtable,A5
          CLR       D4
          MOVE.B    DH7,DL4
          ADD       D4,A5
          MOVE.B    (A5),DL4
          MULU      D0,D4           ; D5 ==> CW*Vn
          MOVE      D5,D4
          ADD       D2,D4           ; D4 = CW(Vn+1.75/11)
          SUB       D2,D5           ; D5 = CW(Vn-1.75/11)
          CMP       A3,D4           ; ck   Vn+1.75/11 < BAR TOTAL
          JMPA      LT,nextcode
          CMP       A3,D5           ; ck   Vn-1.75/11 > BAR TOTAL
          JMPA      GT,nextcode
;*
;* Find the widest bar and space and the narrowest bar and space amoung the
;* six elements making up the current character. If foreward is true, these
;* elements are e1 through e6; if foreward is false, those elements are e2
;* through e7. Next, calculate the ratio's of the widest bar to the narrowest
;* bar and that5 of the widest space to the narrowest space. If either of
;* these ratio's is larger than the maximum element ratio (8.0), quit the
;* decoder (element widths out of tolerance).
;*
c816:     MOVE      iPTR,A5
          CALLA     Ti2
          BTST      #FOREWARD,SR
          JMPR.S    CS,c816_1
          CALLA     Ti2
c816_1:   MOVE      (A5),D1
          MOVE      D1,D3
          CALLA     Ti2
          MOVE      (A5),D2
          MOVE      D2,D4
          MOVE.B    #2,DL7
c816_2:   CALLA     Ti2
          CMP       (A5),D1
          JMPR.S    GE,c816_3
          MOVE      (A5),D1
c816_3:   CMP       (A5),D3
          JMPR.S    LE,c816_4
          MOVE      (A5),D3
c816_4:   CALLA     Ti2
          CMP       (A5),D2
          JMPR.S    GE,c816_5
          MOVE      (A5),D2
c816_5:   CMP       (A5),D4
          JMPR.S    LE,c816_6
          MOVE      (A5),D4
c816_6:   DJNZ.B    DL7,c816_2
c817:     ASL       #3,D3
          CMP       D3,D1            ; wb/nb > 8.0
          JMPA      GT,nextcode
c818:     ASL       #3,D4
          CMP       D2,D4           ; ws/ns > 8.0
          JMPA      GT,nextcode
```

```
;*
;* Find the character width by adding the six elements making up the current
;* character. If foreward is true, those elements are e1 through e6; if fore-
;* ward is false, those elements are e2 through e7. Then, find the bar total
;* and four two term sums of the current character, thusly;
;* If FOREWARD = TRUE                         If FOREWARD = FALSE
;*     T1 = e1+e2                                 T1 = e6+e7
;*     T2 = e2+e3                                 T2 = e5+e6
;*     T3 = e3+e4                                 T3 = e4+e5
;*     T4 = e4+e5                                 T4 = e3+e4
;*     BT = e1+e3+e5/CW                           BT = e3+e5+e7/CW
;*
c811:     MOVE      IPTR,A5
          CLR       A2
          BTST      #FOREWARD,SR
          JMPR.S    CC,c813
c812:     CALLA     T12
          MOVE      (A5),D1
          MOVE      D1,A3
          CALLA     T12
          MOVE      (A5),D2
          ADD       D2,D1          ; T1 = e1+e2
          MOVE      D1,D0
          CALLA     T12
          MOVE      (A5),D3
          ADD       D3,D2          ; T2 = e2+e3
          ADD       D3,A3
          CALLA     T12
          MOVE      (A5),D4
          ADD       D4,D3          ; T3 = e3+e4
          ADD       D3,D0
          CALLA     T12
          ADD       (A5),D4        ; T4 = e4+e5
          ADD       (A5),D0
          ADD       (A5),A3        ; A3 = e1+e3+e5
          CALLA     T12
          ADD       (A5),D0        ; D0 = e1+e2+e3+e4+e5+e6
          JMPR.S    c811a
c813:     CALLA     T14
          MOVE      (A5),D0
          CALLA     T12
          MOVE      (A5),D4
          MOVE      D4,A3
          CALLA     T12
          MOVE      (A5),D3
          ADD       D3,D4          ; T4 = e3+e4
          ADD       D4,D0
          CALLA     T12
          MOVE      (A5),D2
          ADD       D2,A3
          ADD       D2,D3          ; T3 = e4+e5
          CALLA     T12
          MOVE      (A5),D1
          ADD       D1,D2          ; T2 = e5+e6
          ADD       D2,D0
          CALL      T12
          ADD       (A5),D1        ; T1 = e6+e7
          ADD       (A5),D0        ; D0 = e2+e3+e4+e5+e6+e7
          ADD       (A5),A3        ; A3 = e3+e5+e7
c811a:    MOVE      D1,T1
          MOVE      D2,T2
          MOVE      D3,T3
          MOVE      D4,T4          ; AT EXIT  D0 ==> CHAR WIDTH
                                   ;          T1 ==> T1
                                   ;          T2 ==> T2
                                   ;          T3 ==> T3
                                   ;          T4 ==> T4
                                   ;          A3 ==> BAR TOTAL
                                   ;
;*
;* Compute the five threshold values by multiplying thw total character width
;* times the threshold constants 2.5/11, 3.5/11, 4.5/11, 5.5/11, and 6.5/11.
;*
c814:     MOVE      D0,D5
          CLR       D4
          MOVE      #11,D1
          DIVU      D1,D4          ; D5 = CW/11
          MOVE      D5,D1
          LSR       #1,D1
          ADD       D5,D1
          ADD       D5,D1          ; D1 = CW*2.5/11
```

```
        MOVE    D1,D2
        ADD     D5,D2           ; D2 = CW*3.5/11
        MOVE    D2,D3
        ADD     D5,D3           ; D3 = CW*4.5/11
        MOVE    D3,D4
        ADD     D5,D4           ; D4 = CW*5.5/11
        ADD     D4,D5           ; D5 = CW*6.5/11
;*
;* Compute the character pattern. The four digits making up this pattern are
;* generated by doing the following for each two term sum T1 through T4. For
;* j 1 to 4...
;*              Dj = 2 if Tj < thresh1
;*              Dj = 3 if Tj < thresh2
;*              Dj = 4 if Tj < thresh3
;*              Dj = 5 if Tj < thresh4
;*              Dj = 6 if Tj < thresh5
;*              Dj = 7 if Tj >= thresh5
;* the pattern, then, is equal to ==>    d(4) + 16*d(3) + 256*d(3) + 4096*d(1)
;*
c815:   CMP     T1,D1           ; thresh1 > T1
        JMPR.S  LE,c815_1
        MOVE    #$2000,D6
        JMPR.S  c815_6
c815_1: CMP     T1,D2
        JMPR.S  LE,c815_2
        MOVE    #$3000,D6
        JMPR.S  c815_6
c815_2: CMP     T1,D3
        JMPR.S  LE,c815_3
        MOVE    #$4000,D6
        JMPR.S  c815_6
c815_3: CMP     T1,D4
        JMPR.S  LE,c815_4
        MOVE    #$5000,D6
        JMPR.S  c815_6
c815_4: CMP     T1,D5
        JMPR.S  LE,c815_5
        MOVE    #$6000,D6
        JMPR.S  c815_6
c815_5: MOVE    #$7000,D6
c815_6: CMP     T2,D1
        JMPR.S  LE,c815_7
        ADD     #$0200,D6
        JMPR.S  c815_12
c815_7: CMP     T2,D2
        JMPR.S  LE,c815_8
        ADD     #$0300,D6
        JMPR.S  c815_12
c815_8: CMP     T2,D3
        JMPR.S  LE,c815_9
        ADD     #$0400,D6
        JMPR.S  c815_12
c815_9: CMP     T2,D4
        JMPR.S  LE,c815_10
        ADD     #$0500,D6
        JMPR.S  c815_12
c815_10: CMP    T2,D5
        JMPR.S  LE,c815_11
        ADD     #$0600,D6
        JMPR.S  c815_12
c815_11: ADD    #$0700,D6
c815_12: CMP    T3,D1
        JMPR.S  LE,c815_13
        ADD     #$0020,D6
        JMPR.S  c815_18
c815_13: CMP    T3,D2
        JMPR.S  LE,c815_14
        ADD     #$0030,D6
        JMPR.S  c815_18
c815_14: CMP    T3,D3
        JMPR.S  LE,c815_15
        ADD     #$0040,D6
        JMPR.S  c815_18
c815_15: CMP    T3,D4
        JMPR.S  LE,c815_16
        ADD     #$0050,D6
        JMPR.S  c815_18
c815_16: CMP    T3,D5
        JMPR.S  LE,c815_17
        ADD     #$0060,D6
```

```
         JMPR.S     c815_18
c815_17: ADD        #$0070,D6
c815_18: CMP        T4,D1
         JMPR.S     LE,c815_19
         ADD        #$0002,D6
         RET
c815_19: CMP        T4,D2
         JMPR.S     LE,c815_20
         ADD        #$0003,D6
         RET
c815_20: CMP        T4,D3
         JMPR.S     LE,c815_21
         ADD        #$0004,D6
         RET
c815_21: CMP        T4,D4
         JMPR.S     LE,c815_22
         ADD        #$0005,D6
         RET
c815_22: CMP        T4,D5
         JMPR.S     LE,c815_23
         ADD        #$0006,D6
         RET
c815_23: ADD        #$0007,D7
         RET                         ; AT EXIT  D0 ==> CHAR WIDTH
                                     ;          A0 ==> LAST WIDTH
                                     ;          D6 ==> CHAR PATTERN
                                     ;          A3 ==> BAR TOTAL

codeC128: BTST      #C128,DECODER1
          JMPA      CC,nextcode
          MOVE      lastI,A5
          MOVE      A5,IPTR
;*
;* If element(i) < 3*(element(i+1) + element(i+2)) quit, margin width too
;* small.
;*
c803:     MOVE      (A5),D0
          CALLA     T12
          MOVE      (A5),D1
          CALLA     T12
          ADD       (A5),D1
          ADD       D1,D1            ; (e1+e2)*2
          CMP       D0,D1            ; (e1+e2)*2 > e0 ?
          JMPA      GE,nextcode
;*
;* Set foreward true and jump to proceedure c811 and get a character pattern.
;* If the pattern is one of the following, start the label string with that
;* character.
;*                  3255          103 (start A)
;*                  3233          104 (start B)
;*                  3235          105 (start C)
;*                  3224          107 (backward stop)
;* If character is 107 (backward stop) set the foreward flag false. If none
;* of these four charactars is found, quit the algorithm. Other wise check
;* the character pattern parity and widths (c807 and c816). If either of these
;* tests fail, quit. Otherwise, move IPTR one frame_width foreward (6 elem-
;* ents) in the data buffer.
;*
c804:     BSET      #FOREWARD,SR
          CALLA     c811             ; AT EXIT  D0 ==> CHAR WIDTH
          CMP       #$3255,D6        ;          A0 ==> LAST WIDTH
          JMPR.S    NE,c804_1        ;          D6 ==> CHAR PATTERN
          MOVE.B    #103,DH7         ;          A3 ==> BAR TOTAL
          JMPR.S    c804_4
c804_1:   CMP       #$3233,D6
          JMPR.S    NE,c804_2
          MOVE.B    #104,DH7
          JMPR.S    c804_4
c804_2:   CMP       #$3235,D6
          JMPR.S    NE,c804_3
          MOVE.B    #105,DH7
```

```
            JMPR.S    c804_4
c804_3:     CMP       #$3224,D6
            JMPA      NE,nextcode
            MOVE.B    #107,DN7
c804_4:     CLR       LABEL_BUF
            MOVE      #LABEL_BUF+1,A4
            MOVE.B    DN7,(A4)+
            ADD.B     #1,LABEL_BUF
;*
c128loop:   MOVE      IPTR,A5
            ADD       #12,A5            ; increment pointer by 1 frame width
            CMP       #CDEND,A5
            JMPR.S    LT,c128_1
            SUB       #WIDTH,A5
c128_1:     MOVE      A5,IPTR
;*
c805:       CALLA     room
;*
;* Do the procedure at c811 to get a character pattern; then use the lookup
;* table at ref. B.3.6 to get a character. The following tree accomplishes
;* this by looking at the two most significant digits of the character
;* pattern and, if a match is found, attempts to match the two least sign-
;* ificant digits. If no match is found, quit (invalid character).
;*
c806:       CALLA     c811              ; get a character
            CMP.B     #$22,DM6
            JMPR.S    NE,c806_6
            CMP.B     #$25,DL6          ; 2225
            JMPR.S    NEc806_1
            MOVE.B    #92,DN7
            JMPA      c807
c806_1:     CMP.B     #$34,DL6          ; 2234
            JMPR.S    NEc806_2
            MOVE.B    #63,DN7
            JMPA      c807
c806_2:     CMP.B     #$36,DL6          ; 2236
            JMPR.S    NEc806_3
            MOVE.B    #60,DN7
            JMPA      c807
c806_3:     CMP.B     #$45,DL6          ; 2245
            JMPR.S    NEc806_4
            MOVE.B    #33,DN7
            JMPA      c807
c806_4:     CMP.B     #$47,DL6          ; 2247
            JMPR.S    NEc806_5
            MOVE.B    #93,DN7
            JMPA      c807
c806_5:     CMP.B     #$56,DL6          ; 2256
            JMPA      NE,nextcode
            MOVE.B    #64,DN7
            JMPA      c807
c806_6:     CMP.B     #$23,DM6
            JMPR.S    NE,c806_12
            CMP.B     #$34,DL6          ; 2334
            JMPR.S    NE,c806_7
            MOVE.B    #42,DN7
            JMPA      c807
c806_7:     CMP.B     #$43,DL6          ; 2343
            JMPR.S    NE,c806_8
            MOVE.B    #69,DN7
            JMPA      c807
c806_8:     CMP.B     #$45,DL6          ; 2345
            JMPR.S    NE,c806_9
            MOVE.B    #12,DN7
            JMPA      c807
c806_9:     CMP.B     #$54,DL6          ; 2354
            JMPR.S    NE,c806_10
            MOVE.B    #36,DN7
            JMPA      c807
c806_10:    CMP.B     #$56,DL6          ; 2356
            JMPR.S    NE,c806_11
            MOVE.B    #43,DN7
            JMPA      c807
c806_11:    CMP.B     #$65,DL6          ; 2365
            JMPA      NE,nextcode
            MOVE.B    #70,DN7
            JMPA      c807
c806_12:    CMP.B     #$24,DM6
            JMPR.S    NE,c806_16
            CMP.B     #$43,DL6          ; 2443
```

```
          JMPR.S   NE,c806_13
          MOVE.B   #45,DH7
          JMPA     c807
c806_13:  CMP.B    #$45,DL6        ; 2445
          JMPR.S   NE,c806_14
          MOVE.B   #99,DH7
          JMPA     c807
c806_14:  CMP.B    #$54,DL6        ; 2454
          JMPR.S   NE,c806_15
          MOVE.B   #15,DH7
          JMPA     c807
c806_15:  CMP.B    #$65,DL6        ; 2465
          JMPA     NE,nextcode
          MOVE.B   #46,DH7
          JMPA     c807
c806_16:  CMP.B    #$25,DH6
          JMPR.S   NE,c806_20
          CMP.B    #$52,DL6        ; 2552
          JMPR.S   NE,c806_17
          MOVE.B   #95,DH7
          JMPA     c807
c806_17:  CMP.B    #$54,DL6        ; 2554
          JMPR.S   NE,c806_18
          MOVE.B   #100,DH7
          JMPA     c807
c806_18:  CMP.B    #$63,DL6        ; 2563
          JMPR.S   NE,c806_19
          MOVE.B   #83,DH7
          JMPA     c807
c806_19:  CMP.B    #$74,DL6        ; 2574
          JMPA     NE,nextcode
          MOVE.B   #96,DH7
          JMPA     c807
c806_20:  CMP.B    #$32,DH6
          JMPR.S   NE,c806_26
          CMP.B    #$24,DL6        ; 3224
          JMPR.S   NE,c806_21
          MOVE.B   #107,DH7        ; backward stop
          JMPA     c807
c806_21:  CMP.B    #$33,DL6        ; 3233
          JMPR.S   NE,c806_22
          MOVE.B   #104,DH7        ; start B
          JMPA     c807
c806_22:  CMP.B    #$35,DL6        ; 3235
          JMPR.S   NE,c806_23
          MOVE.B   #105,DH7        ; start C
          JMPA     c807
c806_23:  CMP.B    #$44,DL6        ; 3244
          JMPR.S   NE,c806_24
          MOVE.B   #39,DH7
          JMPA     c807
c806_24:  CMP.B    #$46,DL6        ; 3246
          JMPR.S   NE,c806_25
          MOVE.B   #49,DH7
          JMPA     c807
c806_25:  CMP.B    #$55,DL6        ; 3255
          JMPRA    NE,nextcode
          MOVE.B   #103,DH7        ; start A
          JMPA     c807
c806_26:  CMP.B    #$33,DH6
          JMPR.S   NE,c806_36
          CMP.B    #$23,DL6        ; 3332
          JMPR.S   NE,c806_27
          MOVE.B   #65,DH7
          JMPA     c807
c806_27:  CMP.B    #$25,DL6        ; 3325
          JMPR.S   NE,c806_28
          MOVE.B   #81,DH7
          JMPA     c807
c806_28:  CMP.B    #$33,DL6        ; 3333
          JMPR.S   NE,c806_29
          MOVE.B   #30,DH7
          JMPA     c807
c806_29:  CMP.B    #$34,DL6        ; 3334
          JMPR.S   NE,c806_30
          MOVE.B   #03,DH7
          JMPA     c807
c806_30:  CMP.B    #$35,DL6        ; 3335
          JMPR.S   NE,c806_31
          MOVE.B   #89,DH7
```

```
            JMPA      c807
c806_31:    CMP.B     #$36,DL6        ; 3336
            JMPR.S    NE,c806_32
            MOVE.B    #82,DH7
            JMPA      c807
c806_32:    CMP.B     #$44,DL6        ; 3344
            JMPR.S    NE,c806_33
            MOVE.B    #00,DH7
            JMPA      c807
c806_33:    CMP.B     #$45,DL6        ; 3345
            JMPR.S    NE,c806_34
            MOVE.B    #04,DH7
            JMPA      c807
c806_34:    CMP.B     #$55,DL6        ; 3355
            JMPR.S    NE,c806_35
            MOVE.B    #31,DH7
            JMPA      c807
c806_35:    CMP.B     #$56,DL6        ; 3356
            JMPA      NE,nextcode
            MOVE.B    #66,DH7
            JMPA      c807
c806_36:    CMP.B     #$34,DH6
            JMPR.S    NE,c806_46
            CMP.B     #$32,DL6        ; 3432
            JMPR.S    NE,c806_37
            MOVE.B    #71,DH7
            JMPA      c807
c806_37:    CMP.B     #$34,DL6        ; 3434
            JMPR.S    NE,c806_38
            MOVE.B    #13,DH7
            JMPA      c807
c806_38:    CMP.B     #$42,DL6        ; 3442
            JMPR.S    NE,c806_39
            MOVE.B    #51,DH7
            JMPA      c807
c806_39:    CMP.B     #$43,DL6        ; 3443
            JMPR.S    NE,c806_40
            MOVE.B    #06,DH7
            JMPA      c807
c806_40:    CMP.B     #$44,DL6        ; 3444
            JMPR.S    NE,c806_41
            MOVE.B    #53,DH7
            JMPA      c807
c806_41:    CMP.B     #$45,DL6        ; 3445
            JMPR.S    NE,c806_42
            MOVE.B    #14,DH7
            JMPA      c807
c806_42:    CMP.B     #$53,DL6        ; 3453
            JMPR.S    NE,c806_43
            MOVE.B    #21,DH7
            JMPA      c807
c806_43:    CMP.B     #$54,DL6        ; 3454
            JMPR.S    NE,c806_44
            MOVE.B    #07,DH7
            JMPA      c807
c806_44:    CMP.B     #$64,DL6        ; 3464
            JMPR.S    NE,c806_45
            MOVE.B    #52,DH7
            JMPA      c807
c806_45:    CMP.B     #$65,DL6        ; 3465
            JMPA      NE,nextcode
            MOVE.B    #72,DH7
            JMPA      c807
c806_46:    CMP.B     #$35,DH6
            JMPR.S    NE,c806_49
            CMP.B     #$43,DL6        ; 3543
            JMPR.S    NE,c806_47
            MOVE.B    #16,DH7
            JMPA      c807
c806_47:    CMP.B     #$53,DL6        ; 3553
            JMPR.S    NE,c806_48
            MOVE.B    #90,DH7
            JMPA      c807
c806_48:    CMP.B     #$54,DL6        ; 3554
            JMPA      NE,nextcode
            MOVE.B    #17,DH7
            JMPA      c807
c806_49:    CMP.B     #$36,DH6
            JMPR.S    NE,c806_51
            CMP.B     #$52,DL6        ; 3652
```

```
          JMPR.S   NE,c806_50
          MOVE.B   #84,DH7
          JMPA     c807
c806_50:  CMP.B    #$52,DL6        ; 3652
          JMPA     NE,nextcode
          MOVE.B   #85,DH7
          JMPA     c807
c806_51:  CMP.B    #$42,DH6
          JMPR.S   NE,c806_55
          CMP.B    #$23,DL6        ; 4223
          JMPR.S   NE,c806_52
          MOVE.B   #54,DH7
          JMPA     c807
c806_52:  CMP.B    #$25,DL6        ; 4225
          JMPR.S   NE,c806_53
          MOVE.B   #101,DH7
          JMPA     c807
c806_53:  CMP.B    #$34,DL6        ; 4234
          JMPR.S   NE,c806_54
          MOVE.B   #24,DH7
          JMPA     c807
c806_54:  CMP.B    #$45,DL6        ; 4245
          JMPA     NE,nextcode
          MOVE.B   #55,DH7
          JMPA     c807
c806_55:  CMP.B    #$43,DH6
          JMPR.S   NE,c806_65
          CMP.B    #$22,DL6        ; 4322
          JMPR.S   NE,c806_56
          MOVE.B   #76,DH7
          JMPA     c807
c806_56:  CMP.B    #$24,DL6        ; 4224
          JMPR.S   NE,c806_57
          MOVE.B   #19,DH7
          JMPA     c807
c806_57:  CMP.B    #$32,DL6        ; 4232
          JMPR.S   NE,c806_58
          MOVE.B   #57,DH7
          JMPA     c807
c806_58:  CMP.B    #$33,DL6        ; 4233
          JMPR.S   NE,c806_59
          MOVE.B   #09,DH7
          JMPA     c807
c806_59:  CMP.B    #$34,DL6        ; 4234
          JMPR.S   NE,c806_60
          MOVE.B   #23,DH7
          JMPA     c807
c806_60:  CMP.B    #$35,DL6        ; 4235
          JMPR.S   NE,c806_61
          MOVE.B   #20,DH7
          JMPA     c807
c806_61:  CMP.B    #$43,DL6        ; 4243
          JMPR.S   NE,c806_62
          MOVE.B   #27,DH7
          JMPA     c807
c806_62:  CMP.B    #$44,DL6        ; 4244
          JMPR.S   NE,c806_63
          MOVE.B   #10,DH7
          JMPA     c807
c806_63:  CMP.B    #$54,DL6        ; 4254
          JMPR.S   NE,c806_64
          MOVE.B   #58,DH7
          JMPA     c807
c806_64:  CMP.B    #$55,DL6        ; 4255
          JMPR.S   NE,nextcode
          MOVE.B   #61,DH7
          JMPA     c807
c806_65:  CMP.B    #$44,DH6
          JMPR.S   NE,c806_72
          CMP.B    #$23,DL6        ; 4423
          JMPR.S   NE,c806_66
          MOVE.B   #34,DH7
          JMPA     c807
c806_66:  CMP.B    #$25,DL6        ; 4425
          JMPR.S   NE,c806_67
          MOVE.B   #94,DH7
          JMPA     c807
c806_67:  CMP.B    #$33,DL6        ; 4233
          JMPR.S   NE,c806_68
          MOVE.B   #01,DH7
```

```
          JMPA      c807
c806_68:  CMP.B     #$34,DL6        ; 4234
          JMPR.S    NE,c806_69
          MOVE.B    #05,DH7
          JMPA      c807
c806_69:  CMP.B     #$43,DL6        ; 4243
          JMPR.S    NE,c806_70
          MOVE.B    #48,DH7
          JMPA      c807
c806_70:  CMP.B     #$44,DL6        ; 4244
          JMPR.S    NE,c806_71
          MOVE.B    #02,DH7
          JMPA      c807
c806_71:  CMP.B     #$45,DL6        ; 4245
          JMPA      NE,nextcode
          MOVE.B    #35,DH7
          JMPA      c807
c806_72:  CMP.B     #$45,DH6
          JMPR.S    NE,c806_79
          CMP.B     #$32,DL6        ; 4532
          JMPR.S    NE,c806_73
          MOVE.B    #37,DH7
          JMPA      c807
c806_73:  CMP.B     #$34,DL6        ; 4534
          JMPR.S    NE,c806_74
          MOVE.B    #44,DH7
          JMPA      c807
c806_74:  CMP.B     #$42,DL6        ; 4542
          JMPR.S    NE,c806_75
          MOVE.B    #22,DH7
          JMPA      c807
c806_75:  CMP.B     #$43,DL6        ; 4543
          JMPR.S    NE,c806_76
          MOVE.B    #08,DH7
          JMPA      c807
c806_76:  CMP.B     #$52,DL6        ; 4552
          JMPR.S    NE,c806_77
          MOVE.B    #60,DH7
          JMPA      c807
c806_77:  CMP.B     #$53,DL6        ; 4553
          JMPR.S    NE,c806_78
          MOVE.B    #18,DH7
          JMPA      c807
c806_78:  CMP.B     #$54,DL6        ; 4554
          JMPR.S    NE,nextcode
          MOVE.B    #38,DH7
          JMPA      c807
c806_79:  CMP.B     #$46,DH6
          JMPR.S    NE,c806_80
          CMP.B     #$43,DL6        ; 4643
          JMPA      NE,nextcode
          MOVE.B    #47,DH7
          JMPA      c807
c806_80:  CMP.B     #$47,DH6
          JMPR.S    NE,c806_81
          CMP.B     #52,DL6         ; 4752
          JMPA      NE,nextcode
          MOVE.B    #79,DH7
          JMPA      c807
c806_81:  CMP.B     #$52,DH6
          JMPR.S    NE,c806_85
          CMP.B     #$22,DL6        ; 5222
          JMPA      NE,c806_82
          MOVE.B    #97,DH7
          JMPA      c807
c806_82:  CMP.B     #$24,DL6        ; 5224
          JMPA      NE,c806_83
          MOVE.B    #102,DH7
          JMPA      c807
c806_83:  CMP.B     #$33,DL6        ; 5233
          JMPA      NE,c806_84
          MOVE.B    #86,DH7
          JMPA      c807
c806_84:  CMP.B     #$44,DL6        ; 5344
          JMPA      NE,nextcode
          MOVE.B    #98,DH7
          JMPA      c807
c806_85:  CMP.B     #$53,DH6
          JMPR.S    NE,c806_88
          CMP.B     #$23,DL6        ; 5323
```

```
          JMPA      NE,c806_86
          MOVE.B    #25,DH7
          JMPA      c807
c806_86:  CMP.B     #$33,DL6        ; 5333
          JMPA      NE,c806_87
          MOVE.B    #91,DH7
          JMPA      c807
c806_87:  CMP.B     #$34,DL6        ; 5334
          JMPA      NE,nextcode
          MOVE.B    #26,DH7
          JMPA      c807
c806_88:  CMP.B     #$54,DH6
          JMPR.S    NE,c806_94
          CMP.B     #$22,DL6        ; 5422
          JMPA      NE,c806_89
          MOVE.B    #40,DH7
          JMPA      c807
c806_89:  CMP.B     #$24,DL6        ; 5424
          JMPA      NE,c806_90
          MOVE.B    #50,DH7
          JMPA      c807
c806_90:  CMP.B     #$32,DL6        ; 5432
          JMPA      NE,c806_91
          MOVE.B    #28,DH7
          JMPA      c807
c806_91:  CMP.B     #$33,DL6        ; 5433
          JMPA      NE,c806_92
          MOVE.B    #11,DH7
          JMPA      c807
c806_92:  CMP.B     #$42,DL6        ; 5442
          JMPA      NE,c806_93
          MOVE.B    #77,DH7
          JMPA      c807
c806_93:  CMP.B     #$43,DL6        ; 5443
          JMPA      NE,c806_94
          MOVE.B    #29,DH7
          JMPA      c807
c806_94:  CMP.B     #$44,DL6        ; 5444
          JMPA      NE,nextcode
          MOVE.B    #41,DH7
          JMPA      c807
c806_94:  CMP.B     #$55,DH6
          JMPR.S    NE,c806_97
          CMP.B     #$23,DL6        ; 5523
          JMPA      NE,c806_95
          MOVE.B    #67,DH7
          JMPA      c807
c806_95:  CMP.B     #$33,DL6        ; 5533
          JMPR.S    NE,c806_96
          MOVE.B    #$32,DH7
          JMPR.S    c807
c806_96:  CMP.B     #$34,DL6        ; 5534
          JMPR.S    NE,nextcode
          MOVE.B    #68,DH7
          JMPR.S    c807
c806_97:  CMP.B     #$56,DH6
          JMPR.S    NE,c806_100
          CMP.B     #$32,DL6        ; 5632
          JMPR.S    NE,c806_98
          MOVE.B    #73,DH7
          JMPR.S    c807
c806_98:  CMP.B     #$42,DL6        ; 5642
          JMPR.S    NE,c806_99
          MOVE.B    #106,DH7        ; fwd stop
          JMPR.S    c807
c806_99:  CMP.B     #$43,DL6        ; 5643
          JMPR.S    NE,c806_100
          MOVE.B    #74,DH7
          JMPR.S    c807
c806_100: CMP.B     #$63,DH6
          JMPR.S    NE,c806_102
          CMP.B     #$22,DL6        ; 6322
          JMPR.S    NE,c806_101
          MOVE.B    #87,DH7
          JMPR.S    c807
c806_101: CMP.B     #$33,DL6        ; 6333
          JMPA      NE,nextcode
          MOVE.B    #88,DH7
          JMPR.S    c807
c806_102: CMP       #$6423,D6       ; 6423
```

```
          JMPR.S    NE,c806_103
          MOVE.B    #56,DH7
          JMPR.S    c807
c806_103: CMP.B     #$65,DH6
          JMPR.B    NE,c806_106
          CMP.B     #$22,DL6          ; 6522
          JMPR.S    NE,c806_104
          MOVE.B    #78,DH7
          JMPR.S    c807
c806_104: CMP.B     #$32,DL6          ; 6532
          JMPR.S    NE,c806_105
          MOVE.B    #59,DH7
          JMPR.S    c807
c806_105: CMP.B     #$33,DL6          ; 6533
          JMPA      NE,nextcode
          MOVE.B    #75,DH7
          JMPR.S    c807
c806_106: CMP       #7422,D6          ; 7422
          JMPA      NE,nextcode
;*
;* Check for parity error using the character value found as an index into
;* VTABLE. Let the value found be Vn, if Vn+1.75/11 < BAR TOTAL or if
;* Vn-1.75/11 > BAR TOTAL, quit (parity error)
;*
c807:     MOVE      D0,D3
          LSR       #2,D3             ; CW/4
          MOVE      D3,D2
          ADD       D3,D3
          ADD       D2,D3             ; CW*3/4
          ADD       D0,D3             ; CW*1.75
          CLR       D2
          MOVE      #11,A5
          DIVU      A5,D2             ; D2 ==> CW*1.75/11
;*
          MOVE      #vtable,A5
          CLR       D4
          MOVE.B    DH7,DL4
          ADD       D4,A5
          MOVE.B    (A5),DL4
          MULU      D0,D4             ; D5 ==> CW*Vn
          MOVE      D5,D4
          ADD       D2,D4             ; D4 = CW(Vn+1.75/11)
          SUB       D2,D5             ; D5 = CW(Vn-1.75/11)
          CMP       A3,D4             ; ck   Vn+1.75/11 < BAR TOTAL
          JMPA      LT,nextcode
          CMP       A3,D5             ; ck   Vn-1.75/11 > BAR TOTAL
          JMPA      GT,nextcode
;*
;* Find the widest bar and space and the narrowest bar and space amoung the
;* six elements making up the current character. If foreward is true, these
;* elements are e1 through e6; if foreward is false, those elements are e2
;* through e7. Next, calculate the ratio's of the widest bar to the narrowest
;* bar and that5 of the widest space to the narrowest space. If either of
;* these ratio's is larger than the maximum element ratio (8.0), quit the
;* decoder (element widths out of tolerance).
;*
c816:     MOVE      IPTR,A5
          CALLA     T12
          BTST      #FOREWARD,SR
          JMPR.S    CS,c816_1
          CALLA     T12
c816_1:   MOVE      (A5),D1
          MOVE      D1,D3
          CALLA     T12
          MOVE      (A5),D2
          MOVE      D2,D4
          MOVE.B    #2,DL7
c816_2:   CALLA     T12
          CMP       (A5),D1
          JMPR.S    GE,c816_3
          MOVE      (A5),D1
c816_3:   CMP       (A5),D3
          JMPR.S    LE,c816_4
          MOVE      (A5),D3
c816_4:   CALLA     T12
          CMP       (A5),D2
          JMPR.S    GE,c816_5
          MOVE      (A5),D2
c816_5:   CMP       (A5),D4
          JMPR.S    LE,c816_6
```

```
        .   MOVE     (A5),D6
c816_6:     DJNZ.B   DL7,c816_2
c817:       ASL      #3,D3
            CMP      D3,D1           ; wb/nb > 8.0
            JMPA     GT,nextcode
c818:       ASL      #3,D4
            CMP      D2,D4           ; ws/ns > 8.0
            JMPA     GT,nextcode
```

Having described the invention in detail and by reference to the preferred embodiment thereof, it will be apparent that other modifications and variations are possible without departing from the scope of the invention defined in the appended claims.

Claims

1. A method of decoding a binary scan signal consisting of a bit sequence produced by an electro-optical scanning device as the device scans bar code symbols on a label, the bits in said sequence corresponding to light and dark spaces making up said bar code symbols, comprising the steps of:

a.) supplying said binary scan signal to a storage buffer such that said buffer contains a plurality of bits most recently produced by said scanning device,

b.) selecting a portion of said bit sequence which defines a large light space,

c.) subjecting the bits in the sequence following those defining said large light space to a series of tests to determine whether such bits were produced by scanning a bar code symbol which is valid in one or more of several bar codes,

d.) decoding the bar code symbol in the codes in which it is valid,

e.) subjecting the next bits in the sequence to a series of tests to determine whether such bits were produced by scanning a bar code symbol which is valid in any of the bar codes in which the previously decoded bar code symbol is valid,

f.) decoding the bar code symbol in the codes in which it and the previously decoded bar code symbol are valid, and

g.) repeating steps e.) and f.) above until all bar code symbols on the label have been decoded.

2. The method of claim 1, in which said several bar codes include one or more codes selected from the group consisting of Code 3 of 9. Interleaved 2 of 5. Codabar, Code 93, Code 128, and UPC/EAN.

3. The method of claim 1 or 2, in which one of said series of tests is the comparison of the element ratio of the bits being tested with preset minimum and maximum element ratio levels, said element ratio being the ratio of the width of the widest of the dark spaces making up the symbol to the width of the narrowest of the dark spaces making up the symbol.

4. The method of claim 1,2 or 3, in which one of said series of tests is the comparison of the element ratio of the bits being tested with preset minimum and maximum element ratio levels, said element ratio being the ratio of the width of the widest of the light spaces making up the symbol to the width of the narrowest of the light spaces making up the symbol.

5. The method of claims 1,2, 3 or 4, in which one of said series of tests is the comparison of the margin ratio of the bits being tested with preset minimum margin ratio level, said margin ratio being the ratio of the width of the large light space preceding the symbols on a label to the sum of the width of the first several light and dark spaces making up the first symbol adjacent the large light space.

6. The method of claim 1, 2, 3, 4 or 5, in which one of the series of tests is the comparison of the threshold ratio of the bits being tested with a preset with a threshold ratio, said threshold ratio being the ratio of the width of the widest light or dark space making up the symbol to the width of a particular light or dark space within the symbol.

7. The method of any preceding claim, in which one of the series of tests is the comparison of the character ratio of the bits being tested with preset maximum and minimum character ratio levels, said character ratio being the ratio of the sum of the widths of the light and dark spaces making up a symbol to the sum of the widths of the light and dark spaces making up the previous symbol.

POOR QUALITY

8. The method of any preceding claim, in which one of the series of tests is the comparison of the gap ratio of the bits being tested with preset maximum and minimum gap ratio levels, said gap ratio being the sum of the widths of the light and dark spaces making up a symbol to the width of the light space between the symbol and an adjacent symbol.

9. The method of any preceding claim, in which one of the series of tests is the comparison of the maximum narrow element ratio of the bits being tested with a preset maximum narrow element ratio level, said maximum narrow element ratio being the greater of the maximum ratio of the width of the narrowest dark space in a symbol to the width of the narrowest light space in the symbol, or the maximum ratio of the width of the narrowest light space in the symbol to the width of the narrowest dark space in the symbol.

10. The method of any preceding claim, in which step d.) includes the step of checking to determine whether the decoded bar code symbol is a backward or forward start or end symbol prior to subjecting further bits in the sequence to testing and decoding.

11. The method of any preceding claim, in which step g.) includes the step of checking the decoded bar code symbol to insure that it is decoded as a symbol which is one of a valid set of such symbols prior to repeating steps e.) and f.).

12. A method of any preceding claim, further comprising the step of comparison of the margin ratio of the bits in the sequence defining the final symbol with a preset minimum margin ratio level, said margin ratio being the ratio of the width of a large light space following the symbols on a label to the sum of the width of the last several light and dark spaces making up the last symbol adjacent the large light space.

13. The method of any preceding claim, in which at least some of the tests to determine whether the bits in the bit sequence were produced by scanning a bar code symbol which is valid in several codes are performed simultaneously.

14. The method of any preceding claim, in which the tests to determine whether the bits in the bit sequence were produced by scanning a bar code symbol which is valid in several codes are performed sequentially.

15. The method of any preceding claim, in which steps b.) through g.) are performed by a programmed digital computer.

16. The method of any preceding claim, in which one of said series of tests is the comparison of a threshold ratio of the bits being tested with several preset ratios, said threshold ratio being the ratio of the width of two of the spaces making up the symbol to the total width of the symbol.

17. A method of decoding a digital scan signal consisting of a bit sequence produced by an electro-optical scanning device as the device scans bar code symbols on a label, comprising the steps of:

a.) storing the bits of the digital scan signal which have been most recently produced by the electro-optical scanning device;

b.) selecting a portion of said bits which defines a large white space;

c.) subjecting a number of the bits in the sequence following those defining said large light space to a series of tests to determine whether such bits were produced by scanning one or more bar code symbols which are valid in one or more of several bar codes; and

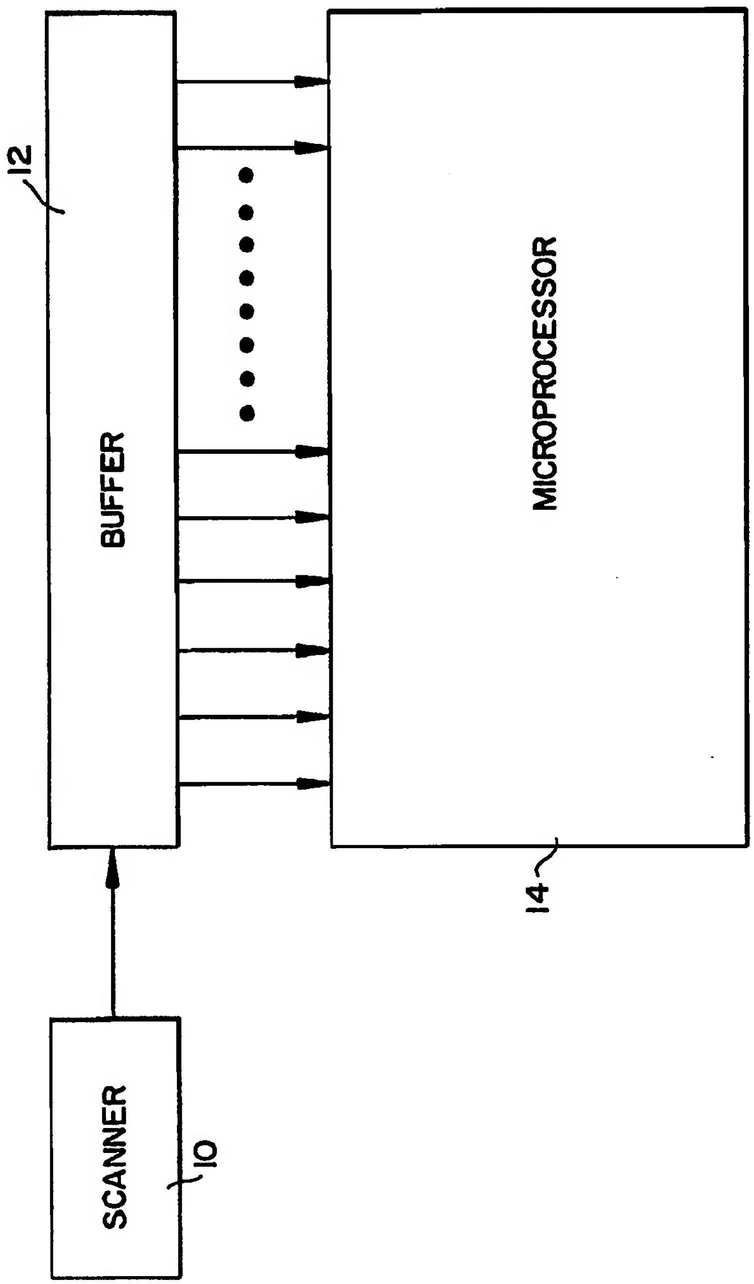
d.) decoding the bits determined to have been produced by scanning one or more bar code symbols which are valid in one or more of said several bar codes.

18. The method of claim 17 for decoding a digital scan signal, further comprising the step of:

e.) performing a series of additional tests on the bits in the decoded sequence to validate that such bits were produced by scanning one or more bar code symbols which are valid in one or more of said several bar codes.

19. The method of claim 17 or 18, in which said several bar codes include one or more codes selected from the group consisting of Code 3 of 9, Interleaved 2 of 5, Codabar, Code 93, Code 128, and UPC/EAN.

20. The method of claim 17, 18 or 19, in which said steps a.) through d.) are performed by a programmed digital computer.


SCANNER
10
BUFFER
12
MICROPROCESSOR
14